

CS51 Assignment 2: Algorithms and Simulation

This assignment has two distinct parts in addition to a reflection: Part 1 requires you to apply your knowledge of algorithmic thinking and optimization and Part 2 allows you to demonstrate your modeling and coding skills by writing a numerical simulation. Material relevant for Part 1 will be covered in class during weeks 3-5, while material relevant for Part 2 will be covered in class during weeks 7 and 8.

You'll notice several "*Optional Challenge*" problems throughout the assignment to challenge yourself. These will only be scored (4 or 5) if they are completed correctly with thorough explanation. If you attempt an optional challenge but do not succeed, you will not be penalized with a low score. Remember that you must include an explanation and interpretation for optional problems to be scored.

This is an individual assignment. We will be checking for similarities among submissions and will take plagiarism seriously.

You must complete all tasks within this pre-formatted Jupyter notebook. Please follow ALL formatting guidelines and the HC Guidelines in the assignment instructions on Forum (near the top and bottom of the instructions respectively).

PART 1: OPTIMIZATION

For this section of the assignment, you will select one of the scenarios below and apply #optimization. You must complete all sections. [#optimization, potentially relevant: #modeling, #algorithms, #variables, #utility, #constraints]

1. *Scenario 1:* To prevent the spread of an infectious disease, a vaccine needs to be distributed as quickly and efficiently as possible to the 15 cities that have had major outbreaks. How can you optimize the route between the cities? For this scenario, you should select cities that are relevant to the disease that you will choose for PART 2. It may be helpful to include a map of these cities (either an existing map or create your own).
2. *Scenario 2:* Suppose that a new virus is starting to spread, and many clinics do not have sophisticated diagnostic tools and must be able to determine whether or not a patient has this dangerous virus based solely on easily measured symptoms. You have been collecting information on symptoms (temperature, WBC count, headache severity, and cough severity) and you need to determine which patients have this new disease and which have only a milder illness. Plots that provide an overview of the data are available [here](https://drive.google.com/file/d/1td_jCTzYiOCMPJ8CwVSJU7u-pQ-5R1JW/view?usp=sharing) (https://drive.google.com/file/d/1td_jCTzYiOCMPJ8CwVSJU7u-pQ-5R1JW/view?usp=sharing). The data can be accessed at [this link](https://drive.google.com/file/d/1smz8OhVklAbWVKpmkx7liE3bOWLfQch7/view?usp=sharing) (<https://drive.google.com/file/d/1smz8OhVklAbWVKpmkx7liE3bOWLfQch7/view?usp=sharing>) (1 = Infected, 0 = Not infected).

1.1 Optimization Problem:

Describe the optimization problem for your scenario: what is the objective function? What are the decision variables? Are there any constraints? Clearly articulate each component so that it's clear how the objective value would be measured and how the decision variables would impact it (~200 words).



Source of image: <https://www.nationsonline.org/oneworld/map/liberia-map.htm>

Scenario 1

Disease: Ebola

The problem is distributing vaccines to 15 cities in Liberia (county capitals) while the objective function is minimizing time taken, as we don't want to give the disease time to spread. I will assume a linear relationship between distance traveled and time taken, allowing the objective function (time in hours) to be a function of distance (meters).

In the real world there might be constraints such as fuel, traversability of path, or available mode of transport. I'll assume neither of these are constraints as including them would add too much complexity, weakening the inferences we can make from the result. The distances between cities would be calculated beforehand. This problem is very similar to the well known traveling salesman problem (TSP), the difference being that we can revisit cities and we also don't care about the journey from the final destination back to the start city. Technically we don't need to create a cycle, just an open path (not returning to start), but for the sake of generality I will create an algorithm for the TSP. The decision variable is the order in which we visit the cities, the path. The path will impact the distance we need to travel, since some paths are longer than others, eg a zig-zag vs square path. The objective function would be measured by adding all the known distances for traveling between cities.

1.2 Optimization Technique:

What process can be used to find the optimal solution in your scenario? Identify and describe an existing algorithm that could be used to complete this process, including the inputs, outputs, required steps, and the termination condition. Explain the advantages and limitations of this algorithm. In your explanation, you should address whether your algorithm would lead to the global optimum and you may wish to compare your algorithm with other possible optimization techniques. (~200 words)

- *Optional:* Draw a flowchart to illustrate the process.
- *Optional challenge:* Create a program in Python to implement this optimization process. Please provide a thorough explanation of how the code works, both holistically and using in-line comments. You must also provide at least one "test case" that demonstrates that your code is properly implemented. This test should be something like, "In this case, it's clear that the only possible solution is X; let's check that with our code. Yes, the code outputs X as well, so we can in principle induce that the implementation is correct."

Ant Colony Optimization (ACO) for the Traveling Salesman Problem-Optimization Problem(TSP-OP)

I'm being specific saying TSP-OP instead of TSP, because TSP could refer to a decision or search problem, for example "Is there a path shorter than x" (YES/NO), or "find a path shorter than x" (PATH). Both of these have results which are trivial to validate. The problem here, TSP-OP, is non-trivial to validate, as knowing if the found path is the shortest is as difficult as the original problem. For TSP, we can't guarantee global minima unless performing a costly a brute force search.

The ACO algorithm is an algorithm that finds near optimal solutions for the TSP-OP. Yet we can't guarantee the global minima. The input for the algorithm is a graph and the weights for each edges, which here will be the euclidian distance between the nodes in the plane. The output will be the path and it's distance. The steps toward the solution can also be seen as an output, as the "certainty" of the algorithm is represented by how quick it finds the solution as well as the presence of multiple paths being marked (the vizualisation of this process can be seen in the animation below)

Defining functions

All code below has been fully created by me and me alone

```
In [1]: 1 # Run this cell to install networkx and plotly library
        2 # Alternatively run "pip install networkx" in your cmd prompt
        3 import sys
        4 !{sys.executable} -m pip install networkx
        5 !{sys.executable} -m pip install plotly
```

Requirement already satisfied: networkx in c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages (2.6.3)

WARNING: Ignoring invalid distribution -lotly (c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages)
WARNING: Ignoring invalid distribution -lotly (c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages)
WARNING: Ignoring invalid distribution -lotly (c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages)
WARNING: Ignoring invalid distribution -lotly (c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages)
WARNING: Ignoring invalid distribution -lotly (c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages)
WARNING: Ignoring invalid distribution -lotly (c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages)
WARNING: You are using pip version 21.3.1; however, version 22.0.3 is available.
You should consider upgrading via the 'c:\users\gabri\appdata\local\programs\python\python39\python.exe -m pip install --upgrade pip' command.

Importing Libraries

```
In [24]: 1  # For Visualizations
2  from plotly.subplots import make_subplots
3  from IPython.display import clear_output
4  from matplotlib.pyplot import figure
5  import plotly.graph_objects as go
6  import matplotlib.pyplot as plt
7  import plotly.express as px
8  from ipywidgets import *
9
10 # Library for manipulating graphs
11 import networkx as nx
12
13 # Functional Libraries
14 import random as rnd
15 import numpy as np
16 import math
17 import time
```

Defining Functions for ACO

In [38]:

```

1  # Create Network
2  def CreateNetwork(amountNodes, createNew):
3      # Create graph, position nodes randomly
4      if createNew:
5          G = nx.complete_graph(amountNodes)
6          _pos = nx.random_layout(G)
7      else:
8          # Set graph and positions to global variables to copy last graph
9          # Used when doing multiple simulations on the same graph
10         G = network
11         _pos = pos
12
13     # Give each edge 2 attributes, weight and pheromone
14     for edge in G.edges():
15         startnode=edge[0]
16         endnode=edge[1]
17
18         if createNew:
19             # Set weight as euclidian distance according to pos
20             distances = round(math.sqrt(((_pos[endnode][1]-_pos[startnode][1])**2)+
21                                     ((_pos[endnode][0]-_pos[startnode][0])**2)),2)
22             # Set pheromones to 1
23             G.add_edge(startnode, endnode, distance=distances, pheromone=1)
24         else:
25             # If not creating new, reset pheromones, distance will already be set
26             G.add_edge(startnode, endnode, pheromone=1)
27
28     # Return the network, and nodes positions
29     return G, _pos
30
31 def GetRandomPathRec(path):
32     # If path contains the whole network
33     # return
34     if len(path) == len(network.nodes):
35         return path
36
37     # Continue from Last node
38     currentNode = path[-1]
39     # Available neighbors
40     possibleNext = []
41     # Neighbors individual probability weight for selection

```



```

42     probArr = []
43
44     # Append each available neighbor
45     for neighbor in list(network[currentNode]):
46         if neighbor not in path:
47             possibleNext.append(neighbor)
48             # Calculate individual prob using TraverseProb()
49             probArr.append(TraverseProb(currentNode, neighbor))
50
51     # Randomly select one neighbor
52     selection = rnd.random() * sum(probArr)
53     currI = -1
54     while selection > 0:
55         currI += 1
56         selection -= probArr[currI]
57
58     # Append selected neighbor
59     path.append(possibleNext[currI])
60
61     # Recursively return the rest of the path
62     return GetRandomPathRec(path)
63
64 def LengthOfPath(path):
65     # Calculate length of path
66     length = 0
67     # Increment with each distance of each edge
68     for i in range(len(path)):
69         length += network.edges[path[i], path[(i + 1) % len(path)]]['distance']
70     return length
71
72 def IncrementPheromones(path, value):
73     # Increment pheromone in each edge from path with value
74     for i in range(len(path)):
75         network.edges[path[i], path[(i + 1) % len(path)]]['pheromone'] += value
76
77 def TraverseProb(nodeA, nodeB):
78     # distance^-distBias * pheromone^pheroBias
79     # (-)distBias because more distance decrease probability
80     # We want bias toward closer nodes
81     return (pow(network.edges[nodeA, nodeB]['distance'], -distBias) *
82             pow(network.edges[nodeA, nodeB]['pheromone'], pheroBias))

```


Defining Functions for visualizations

```
In [39]: 1 # Draw the network
2 def DrawNetwork(pos, weights=[], text=""):
3     # Set size bounds for displaying network
4     figure(figsize=(10, 8), dpi=80)
5
6     # get the phermone of each edge
7     if weights == []:
8         # If weights not given, get weights of current network
9         weights = nx.get_edge_attributes(network, 'pheromone')
10
11     # Width of edge correspond to relative pheromone level
12     widths = np.array(list(weights.values()))
13
14     # Normalize widths
15     widths /= max(widths)
16     widths *= maxThickness
17
18     # Get nodes
19     nodelist = network.nodes()
20
21     # Draw nodes
22     nx.draw_networkx_nodes(network, pos,
23                           nodelist=nodelist,
24                           node_size=300,
25                           node_color='red',
26                           alpha=1)
27
28     # Draw Edges
29     nx.draw_networkx_edges(network, pos,
30                           edgelist = weights.keys(),
31                           width=widths,
32                           edge_color='black',
33                           alpha=1)
34
35     # Draw digits on nodes
36     nx.draw_networkx_labels(network, pos, font_size=10, font_color='white')
37
38     # Update title
39     if text:
40         plt.title(text)
41
42     plt.box(False)
```

```

42     plt.show()
43
44     def DisplayNetworkAnims(pos, index, snapshotInterval, doAnimation=True):
45         # Update function for interactive animation (graph).
46         # I put the Update function inside DisplayNetworkAnims() to get proper variable scope
47         # as this function should only be accesible in this function
48         def Update(step):
49             DrawNetwork(pos, pheroHistories[index][step],
50                         f"Simulation {chr(index + 65)}\nAnts simulated: {step * snapshotInterval}")
51
52         if doAnimation:
53             # Display animation
54             for i in range(len(pheroHistories[index])):
55                 currentAnt = i * snapshotInterval # How many ants have been simulated at snapshot
56                 clear_output(wait=True) # Clear output
57                 DrawNetwork(pos, pheroHistories[index][i], f"Ants simulated: {currentAnt}") # Redraw new graph
58                 time.sleep(.1) # Make it animate slowly
59             time.sleep(1)
60             clear_output(wait=True)
61
62         # interact() creates a interactive slider connected to the fuunction Update
63         print("You can click the slider and then use the left/right arrows to step through the process")
64         interact(Update, step=widgets.IntSlider(min=0, max=len(pheroHistories[index]) - 1, step=1, value=0))
65
66     # Draws a line graph describing the performance of the algorithm
67     def PerformanceGraph(antLen, average, bestLen):
68         # Create subplots depending on how many simulations have been performed (len(antLen) == amount of simulations)
69         # Have 2 graphs per row if simulations > 1
70         fig = make_subplots(rows=math.ceil(len(antLen) / 2), cols=1 if len(antLen) == 1 else 2)
71         avgFig = px.scatter()
72
73         # Boolean for displaying legend
74         displayLegend = True
75
76         for i in range(len(antLen)):
77             # Only display legend for first graph
78             if i == 1:
79                 displayLegend = False
80
81             # Calculate current position among graphs
82             rowPos = (i // 2) + 1
83             colPos = i % 2 + 1

```

```

84
85 # Set titles for each graph
86 fig.update_yaxes(title_text="<b>Length of path</b><br>Unit is (100 km)s", row=rowPos, col=colPos)
87 fig.update_xaxes(title_text="<b>Ants simulated</b>", row=rowPos, col=colPos)
88
89 # Add trace for each Line
90 # Each ants traversed distance
91 fig.add_trace(go.Scatter(y=antLen[i],
92                         mode='lines',
93                         name='Path length for each ant',
94                         line_color='royalblue',
95                         showlegend=displayLegend),
96              row = rowPos, col=colPos)
97
98 # Shortest path so far
99 fig.add_trace(go.Scatter(y=bestLen[i],
100                        mode='lines',
101                        name='Shortest path so far',
102                        line_color='lightgreen',
103                        showlegend=displayLegend),
104             row = rowPos, col=colPos)
105
106 # Critical termination Line at 5% over the shortest path
107 fig.add_trace(go.Scatter(y=np.array(bestLen[i])*(1+terminationLine),
108                        name='5% over shortest path',
109                        line=dict(color='green', dash='dash'),
110                        showlegend=displayLegend),
111             row = rowPos, col=colPos)
112
113 # Filled green area for shortest length
114 fig.add_trace(go.Scatter(
115             x=list(np.array(range(len(bestLen[i])))) +
116                 list(np.array(range(len(bestLen[i])))[::-1]),
117             y=list(np.array(bestLen[i])*(1+terminationLine))+list(np.array(bestLen[i]))[::-1],
118             fill='toself',
119             fillcolor='rgba(107,231,103,0.3)',
120             line_color='rgba(255,255,255,0)',
121             showlegend=displayLegend,
122             name='5% region'),
123             row = rowPos, col=colPos)
124
125 # Running average of traversed distance

```

```

126 fig.add_trace(go.Scatter(x=np.array(range(len(antLen[i]))) + runningAvgOf//2,
127                          y=average[i],
128                          mode='lines',
129                          name=f'Running average of path length (average of {runningAvgOf} ants)',
130                          line_color='red',
131                          showlegend=displayLegend),
132                row = rowPos,
133                col=colPos)
134
135 # Assign letter to each graph
136 fig.add_annotation(text=chr(i + 65),
137                   xref="x domain", yref="y domain",
138                   x=0.95, y=0.95, showarrow=False, row = rowPos, col=colPos, font_size=25)
139
140 # Only do running average graph if there are multiple simulations to compare
141 if len(average) > 1:
142     # avgFig is graph displaying each graphs running average
143     avgFig.add_trace(go.Scatter(x=np.array(range(len(antLen[i]))) + runningAvgOf//2,
144                                y=average[i],
145                                mode='lines+text',
146                                name=f'Sim {chr(i + 65)} running average'))
147
148 # Layout settings for performance graphs
149 fig.update_layout(height=400*(math.ceil(len(antLen) / 2) + 1),
150                  width=1000,
151                  legend=dict(x=0, yanchor="bottom", y=-.15),
152                  title_text=f'''<b>Linegraph showing performance of algorithm</b><br>
153                             Distance of paths over amount of ants simulated''')
154 fig.show()
155
156 # Only do running average graph if there are multiple simulations to compare
157 if len(average) > 1:
158     # Set titles for running average graph
159     avgFig.update_yaxes(title_text=f"<b>Running average of paths</b><br>Unit is (100 km)s")
160     avgFig.update_xaxes(title_text=f"<b>Ants simulated</b>")
161     avgFig.update_layout(title_text=f'''<b>Linegraph showing running average of path length
162                                     over amount of ants simulated</b>''')
163
164     avgFig.show()

```

Algorithm

```
In [40]: 1 # Function for running ACO algorithm
2 def RunACO(nodesAmount,
3             distBias,
4             pheroBias,
5             evaporationRate,
6             pheromoneRate,
7             simulations = 1,
8             newNetwork=True):
9
10     # It is in general a bad practice to make variables global,
11     # as that might lead to unintended access/interactions with the variables,
12     # which can lead to logical errors which are difficult to find.
13     # I still choose to globalize these variables as to
14     # avoid being forced to tunnel them to the functions as parameters,
15     # this increases the readability of the code
16     global network
17     global pos
18
19     # These lists continuously store the relevant data for analysing the performance of the algorithm
20     # I globalize these variables for same reason as stated above.
21     global pheroHistories
22     global maxThickness
23     global antLenHistory
24     global bestLenHistory
25     global pheroHistory
26     global runningAvg
27
28     pheroHistories = []
29     globAntLenHistory = []
30     globBestLenHistory = []
31     globRunningAvgHistory = []
32     bestLengths = []
33     bestPaths = []
34
35     # Defining visual parameters
36     # All data later visualized are stored every 'snapshotInterval' ant
37     snapshotInterval=50
38     # Max thickness of the lines in the network
39     maxThickness = 8
40     # Set fontsize for network visualization
41     plt.rcParams.update({'font.size': 20})
```



```

42
43 # Globalized termination line for same reason as stated above
44 global terminationLine
45 # When the running average distance of the ants hit the line which is
46 # 'terminationLine' % above the shortest distance, it terminates. (bestL * (1+terminationLine))
47 terminationLine = .05
48
49 # Set minimum amount of ants to simulate, in the beginning of the simulation,
50 # the termination condition might be fulfilled, as the best path so far can be very high.
51 # Therefore we don't want to terminate if less than 'minimumAnts' have been simulated
52 minimumAnts = 100
53 # If the algorithm doesn't converge, we wish to terminate after 'maxAmountAnts' no matter what
54 maxAmountAnts = 10000
55
56 # A runningAvgOf 50 means that each value in the list runningAvg is the average of the 50 preceding values
57 global runningAvgOf
58 runningAvgOf=50
59
60 # For each simulation
61 for simiteration in range(simulations):
62
63     # The current 'runningAvgOf' values to average
64     # Works like a que, will be updated to always contain the most
65     # recent 'runningAvgOf' values from antLenHistory
66     currentFocus = []
67
68     # Storing pheromone values
69     pheroHistory = []
70     # Storing length of best path so far
71     bestLenHistory = []
72     # Storing length of each path created
73     antLenHistory = []
74     # Storing the running average of path lengths
75     runningAvg = []
76     # Store what the shortest path is
77     # Stores the order of the cities, listed by their index
78     bestPath = []
79
80     # Create network, only create new network first simulation
81     network, pos = CreateNetwork(nodesAmount, simiteration == 0)
82
83     # Set best length as large

```

```
84 bestL = 1000
85 for i in range(maxAmountAnts + 1):
86
87     # Evaporate pheromones each iteration
88     for edge in network.edges():
89         network.edges[edge]['pheromone'] *= (1 - evaporationRate)
90
91     # Put ant at random node
92     # Make ant go random path
93     path = GetRandomPathRec([rnd.randrange(nodesAmount)])
94     pathL = LengthOfPath(path)
95
96     # Fill up currentFocus until 'runningAvgOf' is reached
97     if i < runningAvgOf:
98         currentFocus.append(pathL)
99     else:
100         # After the first iterations
101         # Add the average to runningAvg
102         runningAvg.append(sum(currentFocus)/len(currentFocus))
103         # Add the new value to currentFocus
104         currentFocus.append(pathL)
105         # Remove the oldest value
106         currentFocus = currentFocus[1:]
107
108         # If the difference between the current running average and the bestlength is
109         # within the termination percentage (5% in this case),
110         # and we've done at least 'minimumAnts' iterations,
111         # Terminate
112         if (runningAvg[-1] - bestL) / bestL < terminationLine and i > minimumAnts:
113             break
114
115     # If current path is shorter than shortest path
116     if pathL < bestL:
117         # Store new best path
118         bestL = pathL
119         bestPath = path
120
121     # Append current values to history lists
122     bestLenHistory.append(bestL)
123     antLenHistory.append(pathL)
124
125     # Store snapshot of pheromones at intervals
```

```

126         if i % snapshotInterval == 0:
127             pheroHistory.append(nx.get_edge_attributes(network, 'pheromone'))
128
129             # diff, how close to best seen solution
130             # Adding .001 to avoid division by zero
131             diff = pathL - bestL + .001
132             IncrementPheromones(path, pheromoneRate/diff)
133
134             # Store each history List in lists,
135             # So we can draw performance graphs on each simulation's history
136             globAntLenHistory.append(antLenHistory)
137             globBestLenHistory.append(bestLenHistory)
138             globRunningAvgHistory.append(runningAvg)
139
140             pheroHistories.append(pheroHistory)
141             bestLengths.append(round(bestL, 6))
142             bestPaths.append(bestPath)
143
144             # Only do the animation if we're doing a single simulation
145             if simulations == 1:
146                 DisplayNetworkAnims(pos, 0, snapshotInterval, True)
147
148             # If simulating multiple times, draw each network, no animation
149             if simulations != 1:
150                 for index in range(len(pheroHistories)):
151                     DisplayNetworkAnims(pos, index, snapshotInterval, False)
152
153             # Draw the performance graph, for each simulation
154             PerformanceGraph(globAntLenHistory,
155                             globRunningAvgHistory,
156                             globBestLenHistory)
157
158             # Return the best paths and their lengths
159             return bestLengths, bestPaths

```

Function calls

In [41]:

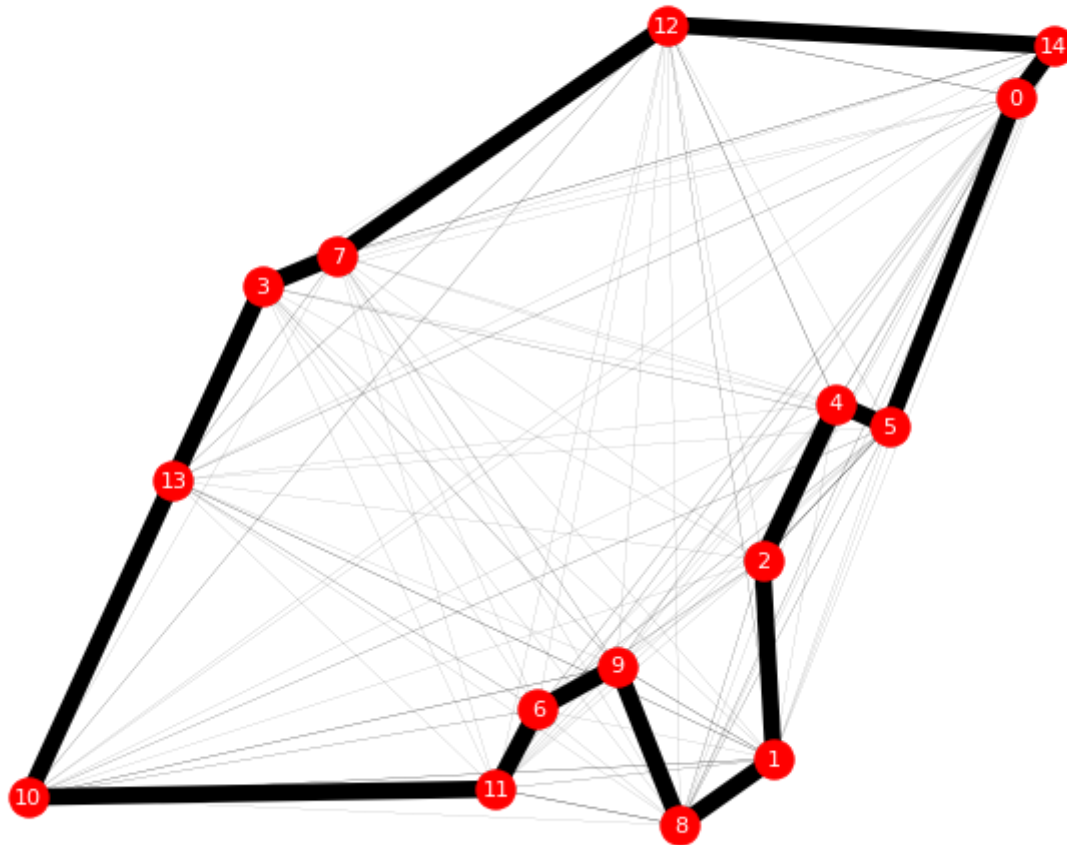
```
1  #Hyperparameters
2  nodesAmount=15
3
4  # Bias variables toward greedy and pheromone approach
5  distBias=2 # Increases bias toward choosing closer nodes
6  pheroBias=1 # Increase bias toward choosing pheromones, edges which have been traversed previously
7
8  # percentage of pheromones that evaporates each iteration
9  evaporationRate = .0001
10
11 # Coefficient for adding pheromones
12 pheromoneRate = .001
13
14 # Run this to see animation and one simulation
15 RunACO(nodesAmount,
16         distBias,
17         pheroBias,
18         evaporationRate,
19         pheromoneRate, 1)
```

You can click the slider and then use the left/right arrows to step through the process

step  11

Simulation A

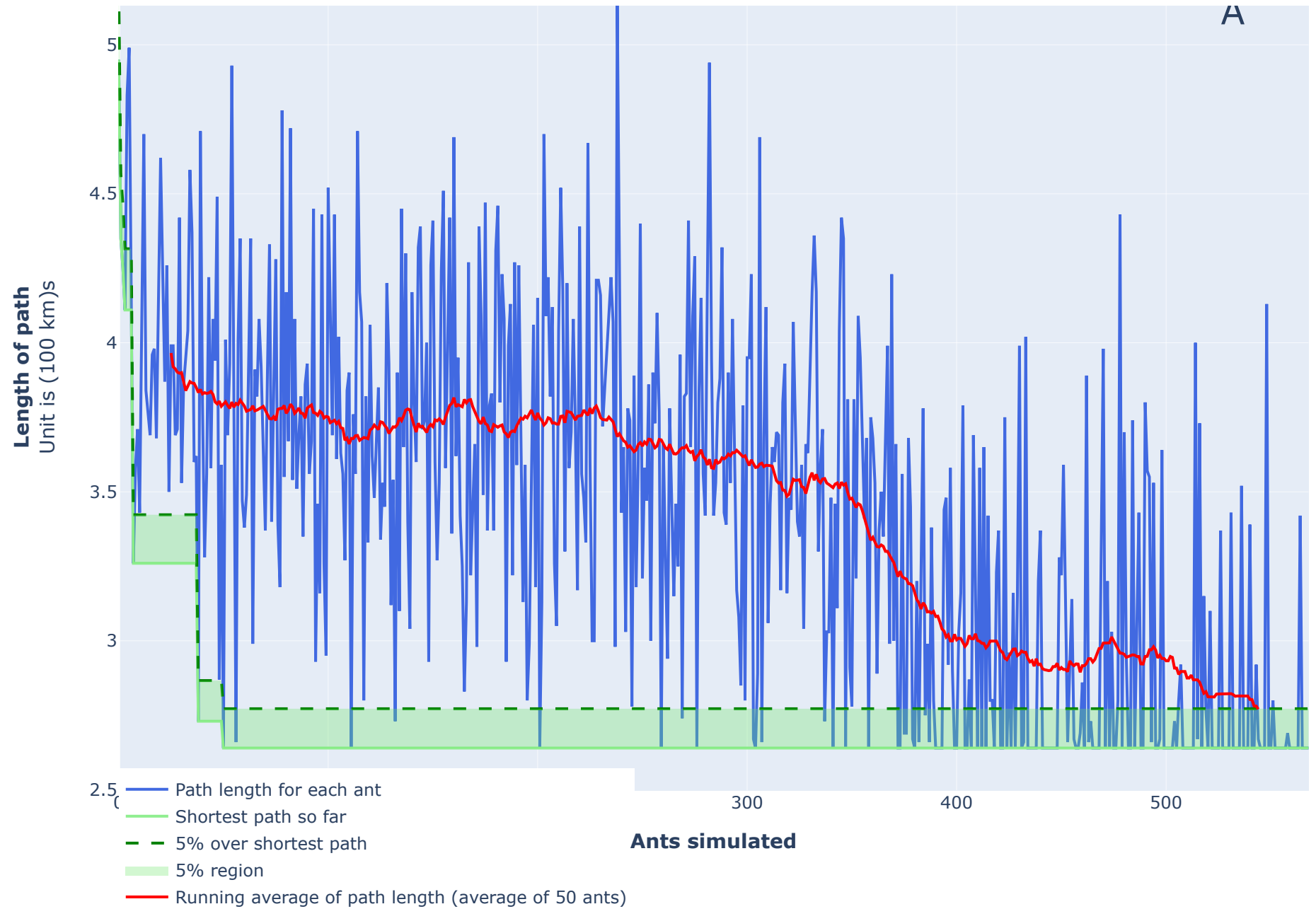
Ants simulated: 550



Linegraph showing performance of algorithm

Distance of paths over amount of ants simulated





Out[41]: ([2.64], [[5, 4, 2, 1, 8, 9, 6, 11, 10, 13, 3, 7, 12, 14, 0]])

In [29]:

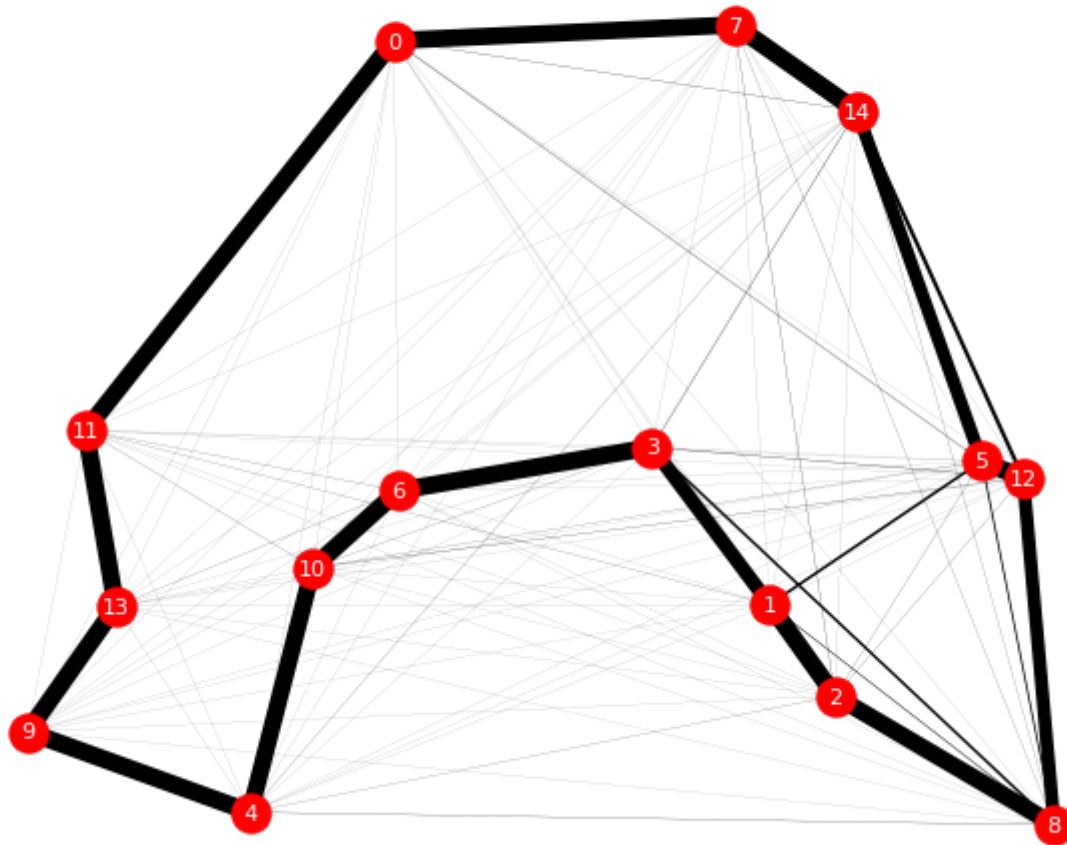
```
1 # How many simulations to run on the same graph
2 simAmount = 6
3
4 # Run this to see multiple simulations
5 RunACO(nodesAmount,
6         distBias,
7         pheroBias,
8         evaporationRate,
9         pheromoneRate, simAmount)
```

You can click the slider and then use the left/right arrows to step through the process

step  16

Simulation A

Ants simulated: 800

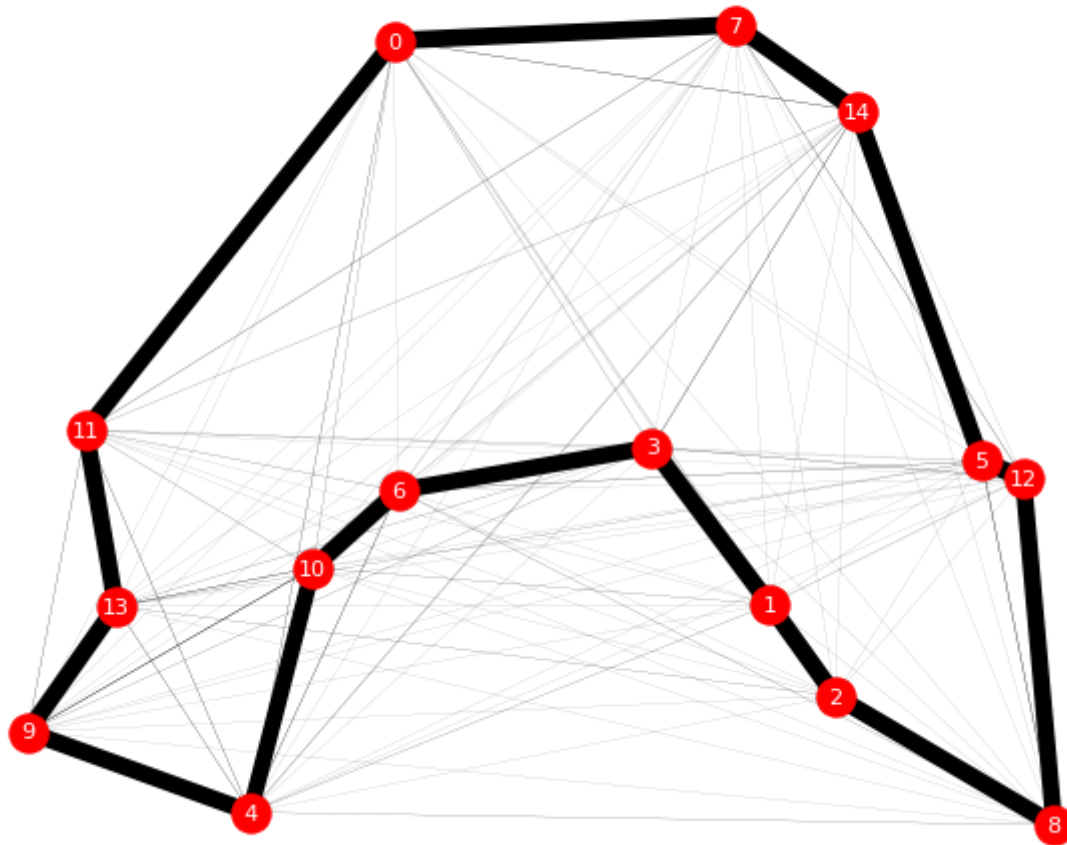


You can click the slider and then use the left/right arrows to step through the process

step 21

Simulation B

Ants simulated: 1050

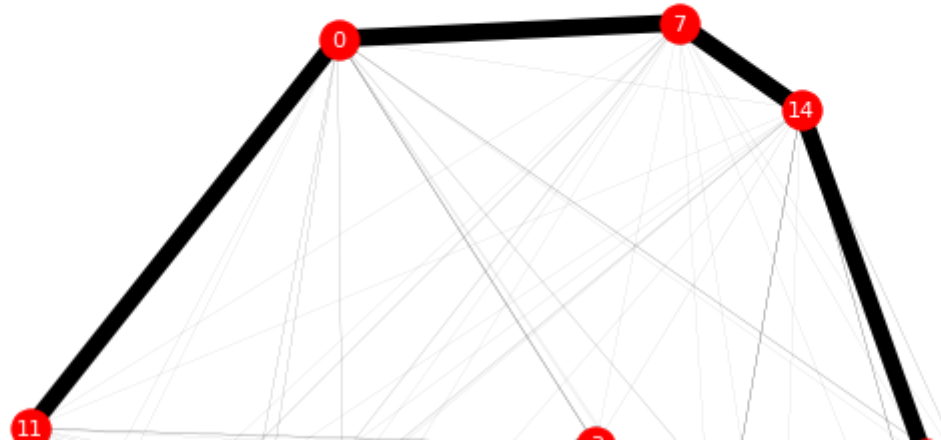


You can click the slider and then use the left/right arrows to step through the process

step  21

Simulation C

Ants simulated: 1050

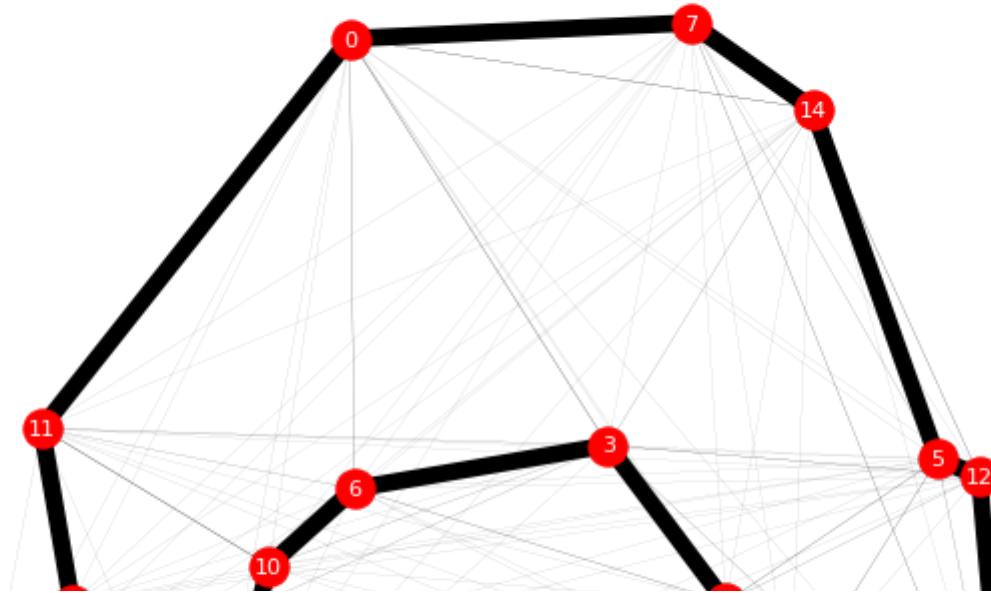


You can click the slider and then use the left/right arrows to step through the process

step 23

Simulation D

Ants simulated: 1150

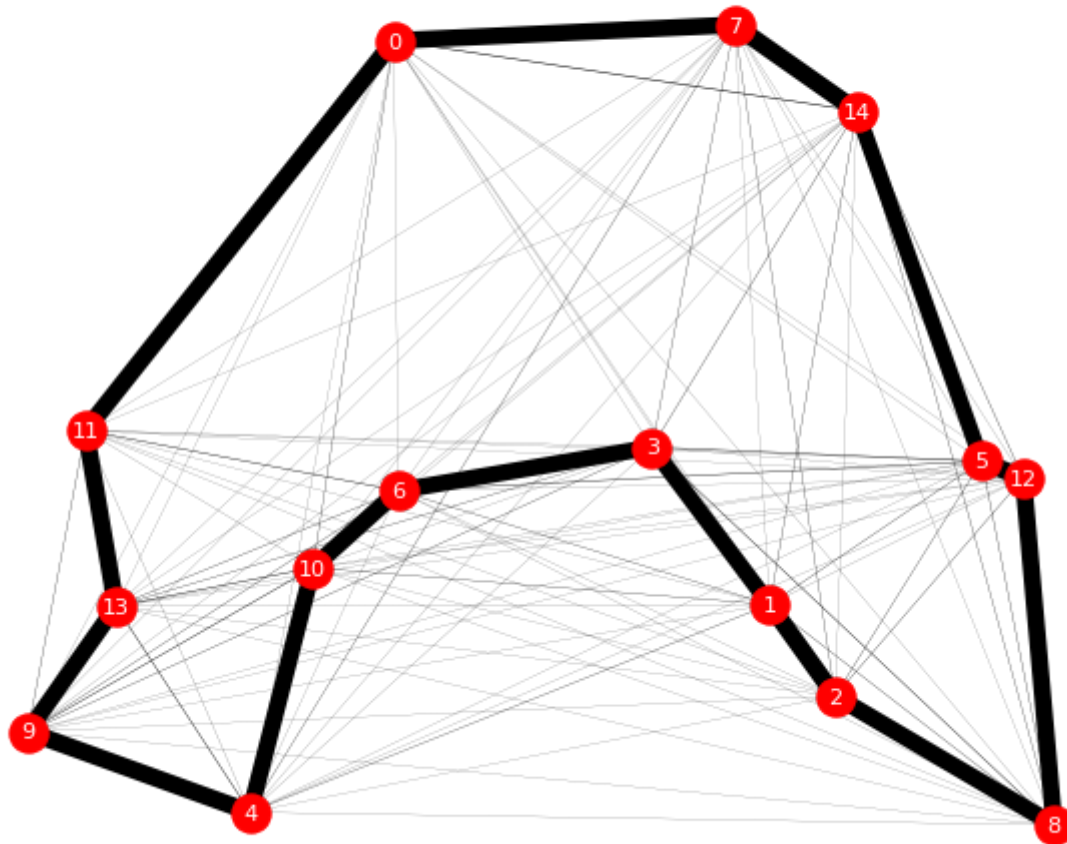


You can click the slider and then use the left/right arrows to step through the process

step 19

Simulation E

Ants simulated: 950

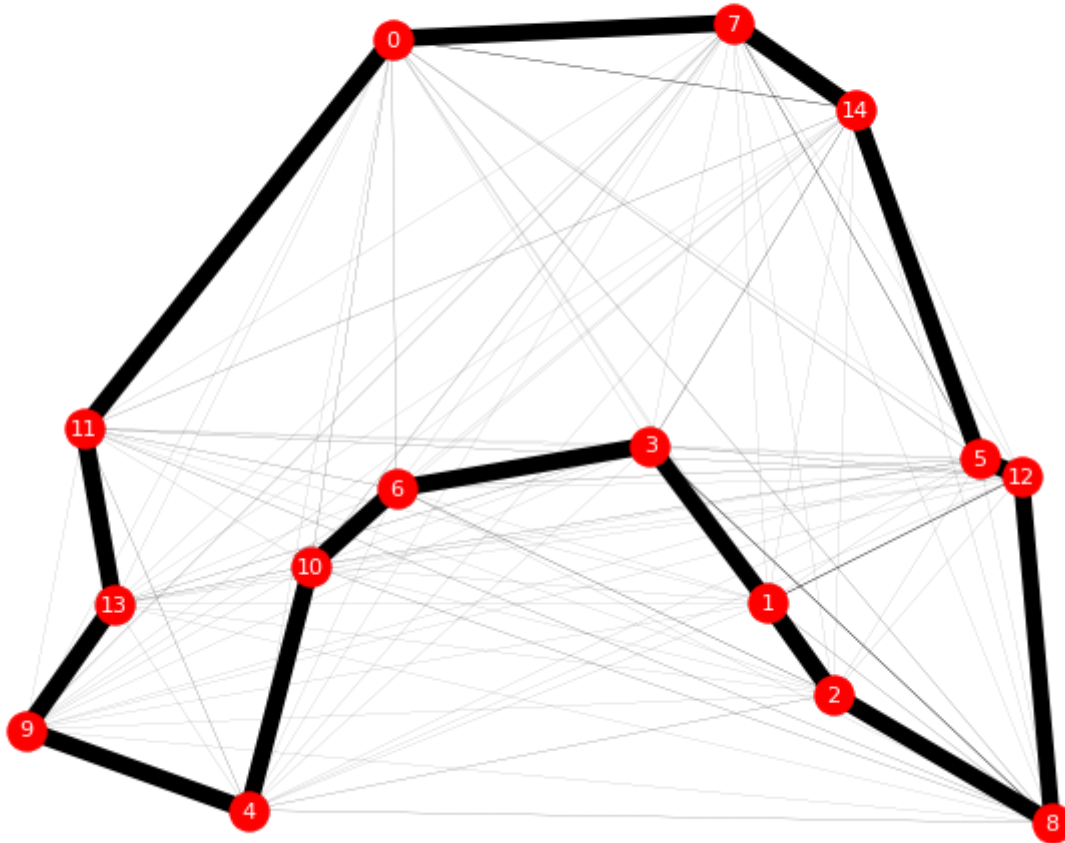


You can click the slider and then use the left/right arrows to step through the process

step 11

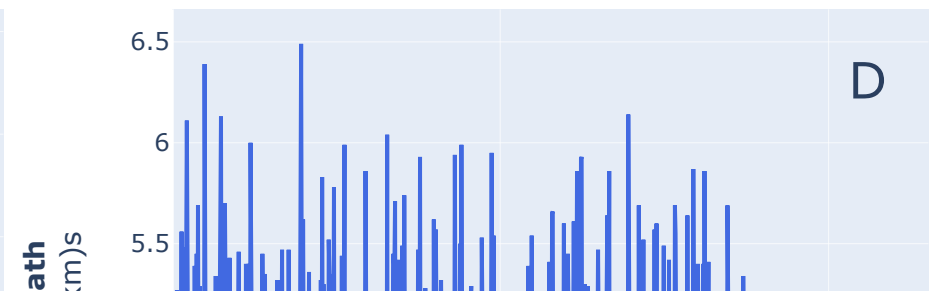
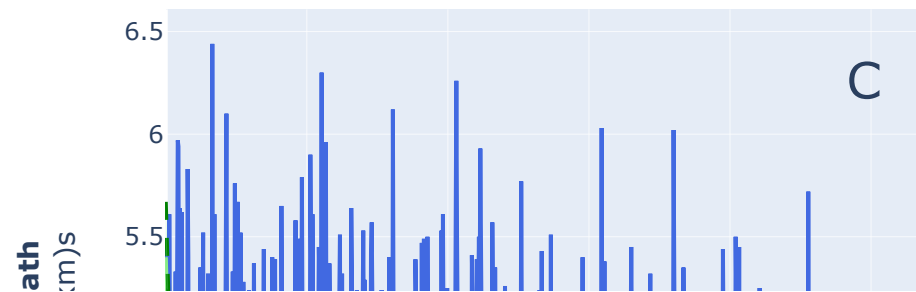
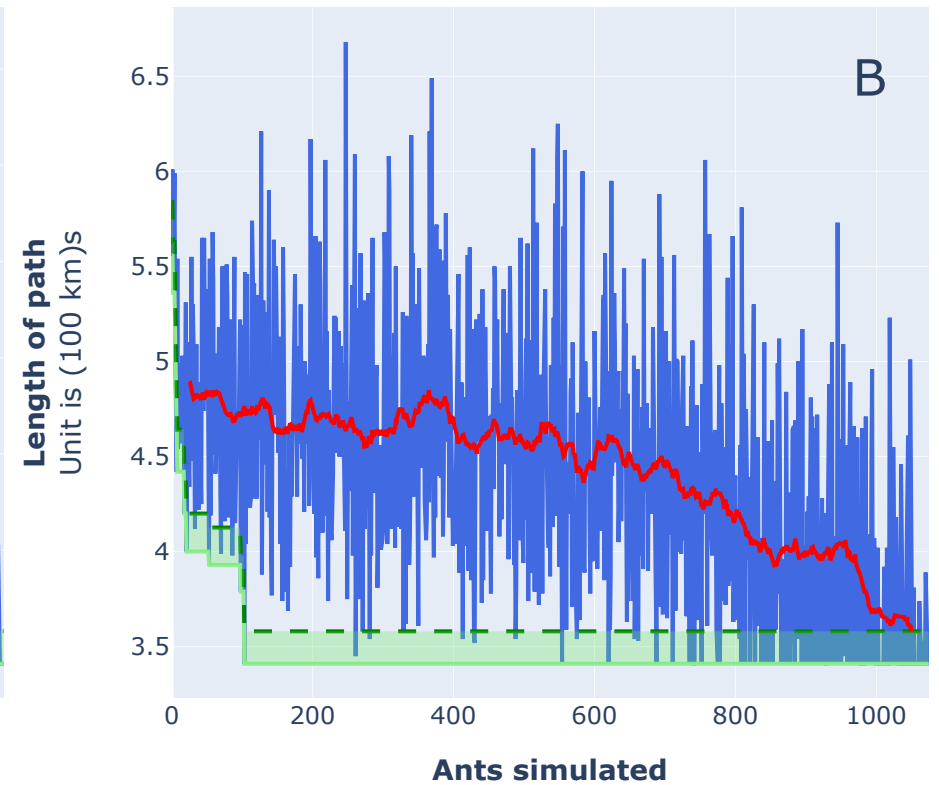
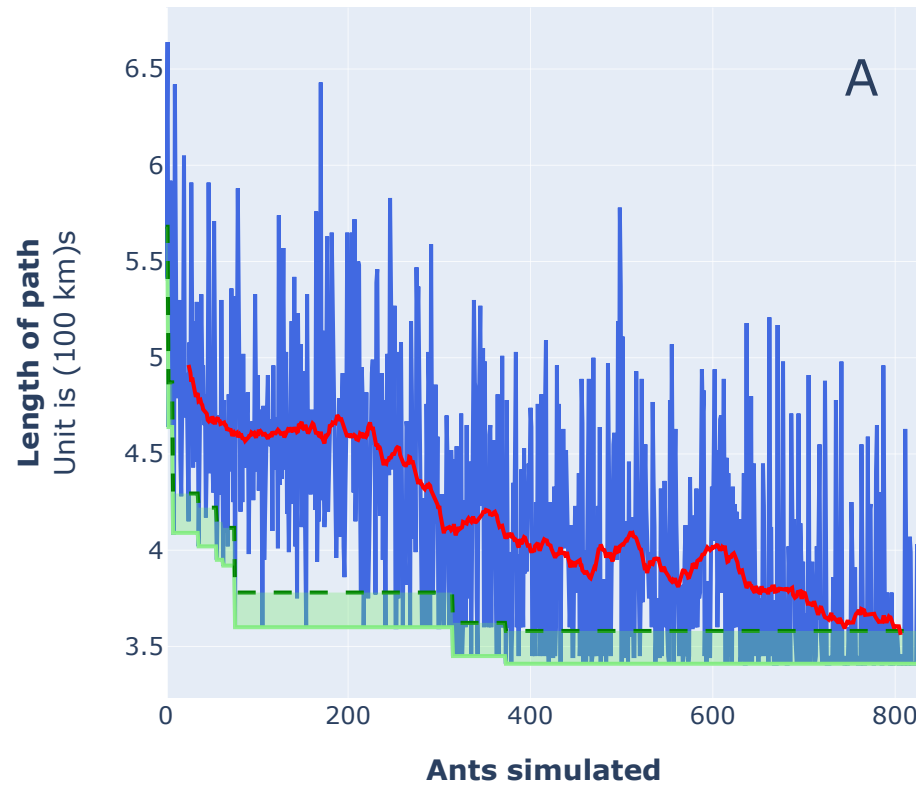
Simulation F

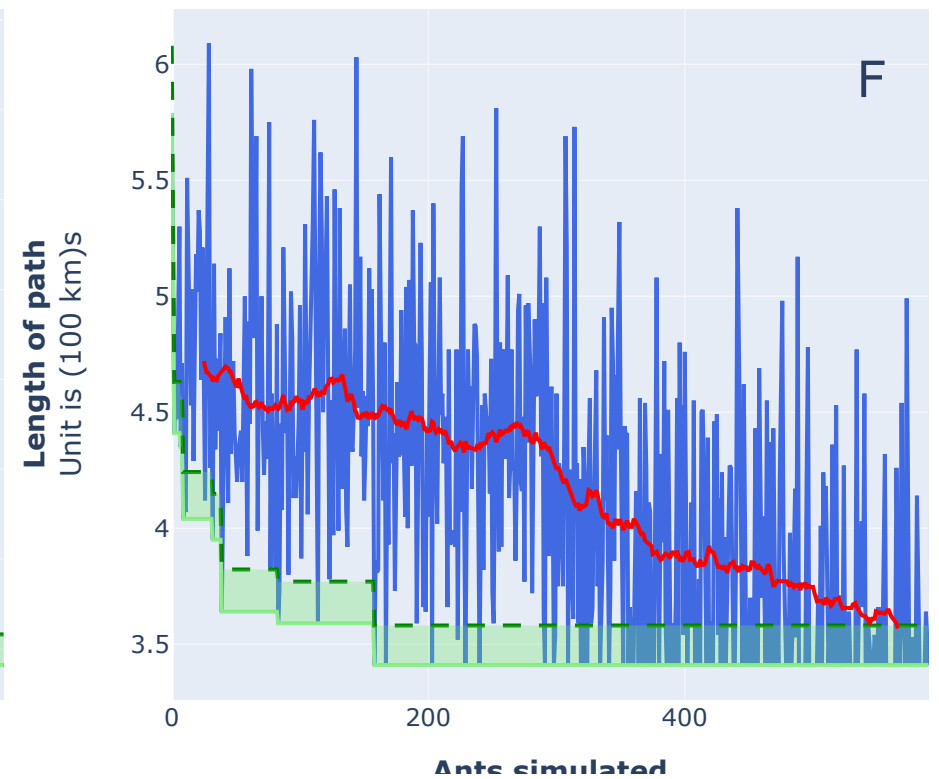
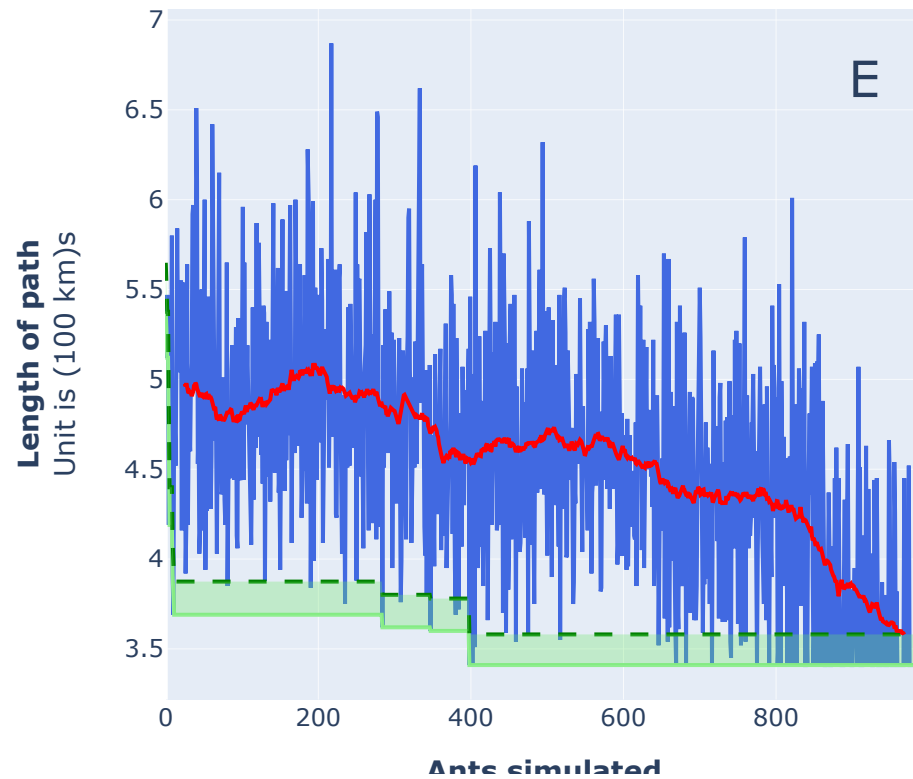
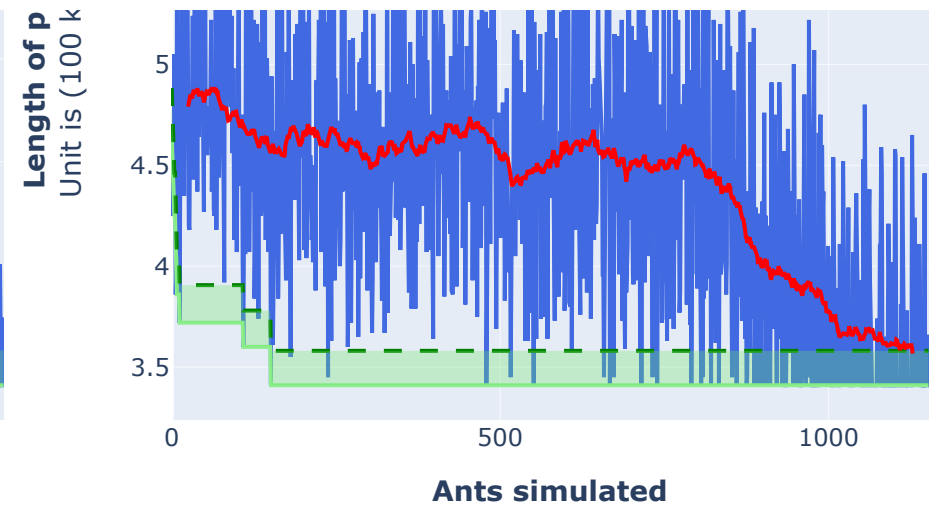
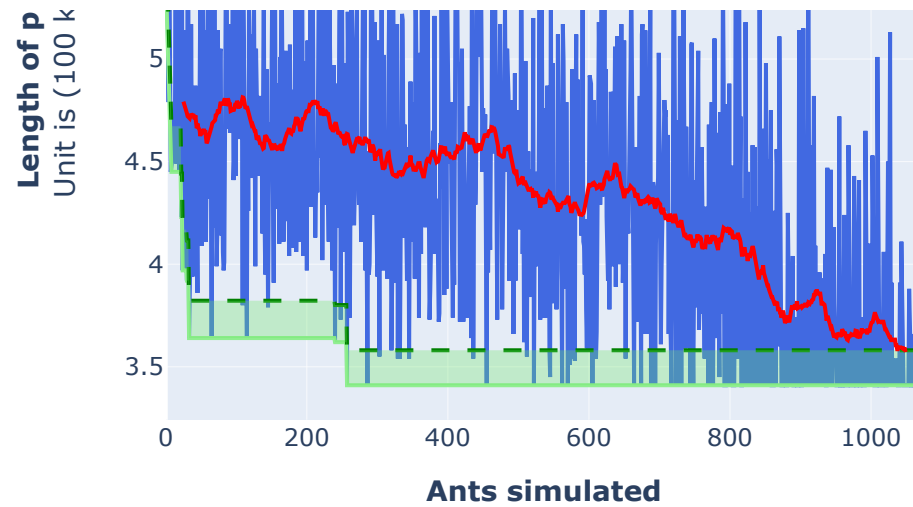
Ants simulated: 550



Linegraph showing performance of algorithm

Distance of paths over amount of ants simulated





Ants simulated

Ants simulated

- Path length for each ant
- Shortest path so far
- 5% over shortest path
- 5% region
- Running average of path length (average of 50 ants)

Linegraph showing running average of path length

over amount of



```
Out[29]: ([3.41, 3.41, 3.41, 3.41, 3.41, 3.41],
[[6, 3, 1, 2, 8, 12, 5, 14, 7, 0, 11, 13, 9, 4, 10],
 [6, 3, 1, 2, 8, 12, 5, 14, 7, 0, 11, 13, 9, 4, 10],
 [6, 3, 1, 2, 8, 12, 5, 14, 7, 0, 11, 13, 9, 4, 10],
 [8, 2, 1, 3, 6, 10, 4, 9, 13, 11, 0, 7, 14, 5, 12],
 [6, 3, 1, 2, 8, 12, 5, 14, 7, 0, 11, 13, 9, 4, 10],
 [6, 3, 1, 2, 8, 12, 5, 14, 7, 0, 11, 13, 9, 4, 10]])
```

[#dataviz]

When displaying the performance of the algorithm I display multiple different graphs showing different elements which are relevant for the performance. For example the interactive animation for the graph displays relevant information about how certain the the algorithm is in the different paths over time. The graphs are informative yet easy to follow with correct labeling and axis names. The graphs allow for focusing at one line at a time and zooming into critical moments, allowing for unique insight about how the algorithm works in its optimization. For example plotting 'individual ant distance' together with 'shortest path so far' gave valuable insight about how the higher level effect of the algorithm 'deciding on a path' can be measured.

[#communicationdesign]

In the animated network graph, I effectively use the principles of salience and informative change. As I adjust the salience of the edges by changing their thickness, this shifts the readers focus and intuitively shows the relevant information about the progress of the algorithm. The color difference between the nodes and edges also increases the discriminability between the relevant components of the graph. The change in edge thickness is a good application of informative change, as the visual salience of them are proportionally representative of how much their included in the probability calculations for the ant. This design grants very intuitive interpretation of a critical component of the algorithm

Notes about output

Observe that we can see the length of the best path at the bottom of the output. When running multiple simulations we can see the stochastic component of this algorithm, as the final solution and the iterations required to get there varies for different smulatons on the same graph. We can see that the simulations that find a suboptimal solution also tend to be the ones that terminated early.

All line graphs above are interactive, you can select and deselect lines by clicking them in the legend. By clicking and dragging inside the graphs you can zoom in.

Explaining the algorithm

The algorithm simulates ants and how they in the real world find the best using pheromones. The concentration of pheromones is an emergent property of accumulated knowledge in the system. The core mechanic that makes this pathfinding work is that shorter paths will accumulate more pheromones as ants will walk there more frequently.

The algorithm works as follows:

Initialization:

1. Set each edge's distance attribute to the euclidian distance between its two nodes
2. Set each edge's pheromone value to 1
3. Define termination condition, maximum and minimum amount of ants to simulate

Algorithm

Foreach ant:

1. Put the ant at a random node
2. The ant chooses the next node to traverse depending on each edge's:
 - A. **Distance attribute**, representing the euclidian distance to that node
 - B. **Pheromone concentration**, representing how much this edge has been traversed and how good(short) the path it resulted in was
 - C. The ant takes these two values to the power of their bias (how biased the ant should be toward each value), and then multiplies the result.

$$distance^{-distBias} \times pheromone^{pheroBias}$$

This value will be calculated for each available node, and is the weighted probability of that node being selected.
3. Select a node randomly depending on their weighted probability. The ant moves to the selected node
4. Repeat 2-3 until all nodes are visited
5. Calculate the length of the ant's path and the difference between this path and the shortest path found so far (save this path length as shortest path so far, if that's the case)
6. Increase the pheromone of each edge traversed by the ant, increase with reciprocal of the difference from step 5 (a small difference would be a good path, which should increase the pheromone more)
7. Take a new ant and go to step 1, repeat until out of ants

Termination condition

There are many ways one could design a termination condition. The goal of my termination condition is to correctly detect when the algorithm has "decided" upon a path, which I define as the probability of the algorithm "changing its mind" is decreasing and accelerating heavily. This event is very clear visibly in my animation, as the chosen paths edges become thicker, and all other edges practically disappear.

This is a good termination condition as running the simulation further would by definition not give a new solution, but only reinforce the one found. A negative is that we can't confirm that this will happen within a reasonable amount of time, for this reason I implemented a maximum amount of ants to simulate, as a secondary termination condition. This hyperparameter, much like the other hyperparameters, are dependent on the amount of nodes in the network, the computational time that is reasonable for the context, and the desired confidence.

When deciding on a termination condition I utilized the linegraph I produced which described the progress of the ants, this graph gave valuable insight on how to detect this event. First I considered only running for a predetermined amount of iterations only, which could be a sufficient condition depending on context, but clearly not effective as it sometimes would perform computation that has no effect on the result, and sometimes it would terminate before a path had been clearly decided upon. After examining the graph I noticed that the distribution of path-distances was "cut-off" approximately at the same time as when the desired effect could be visually seen in the animation. From this I added both the red and green line, the average distance of the paths and the shortest path found so far. With those additions it became apparent how the "deciding" event occurred when a more significant portion of the ants took the best path found so far, which further reinforces it and further increases the amount of ants taking that path. When formulating the termination condition I considered:

- "when x ants in a row go on the best path", but I concluded that was too affected by random events
- "When the average range of the distribution of path-distances was cut to 75%", but it would be difficult to define and difficult to interpret how that confidently is connected to a decision being made
- "When ants taking the best path represent the fraction x of the latest n ants", I felt this would be too affected by randomization and it'd be unclear how to specify these new hyperparameters Finally I stopped on
- "When the running average (last 50 ants) of the distance is within 5% of the shortest path", This is an accurate proxy for the decision event, as well as being easy to implement, interpret, and cheap to check for. It's also clear how this is directly connected to more ants taking the shortest path and starting the reinforcement loop. This critical line is represented by the green dashed line in the linegraph.

Another termination condition that would be a more accurate representation of "one single path being chosen" is to see if the edges contained in the path are the only edges which are "activated", where not activated would be represented by edges having pheromone levels below some critical level. Where that critical level would be so low compared to the activated edges that they're effectively irrelevant. This would be a good termination condition, but I judged it to be too costly to compute in each iteration, as I would have to check the levels of each edge every single iteration

Pheromone evaporation

The evaporation rate of the system is proportional to the current pheromone level, while the incrementation of pheromones is a constant value. This means that when the current pheromone levels are low, the impact of increasing pheromones will be relatively larger than the rate of evaporation.

Vice versa is also true, if the pheromone levels are large the evaporation will be larger as well. Practically this means that an edge with less

pheromones will, after being visited, get a more longer lasting impact than an edge that already has a lot of pheromones, there is a bias towards edges that are not visited that much, promoting any shorter paths found by random exploration. This effect is a balance against the effect that paths with more pheromones will get visited a lot more. If the evaporation was not proportional, but had a constant value for decrementation, it would increase the probability that the algorithm would reinforce early solutions too heavily and settle on a local minima.

Discussion

ACO is a stochastic optimization algorithm which, similar to simulated annealing, can find the global minima by allowing for random and temporary decrease in the objective function to fall into other basins that might lead to a better local minima, or perhaps even global minima (assuming we're looking to maximize). This is preferable over greedy and naive algorithms which have no process to avoid getting stuck in a local minima.

This stochastic property of course leaves the possibility for varying results on the same graph, therefore introducing the probability of suboptimal solutions. In ACO there are many small entities, ants, having a small impact on the system probabilities, pheromones. This in combination with the fact that better paths gets more pheromone than worse paths means that the only way for a suboptimal path to get chosen is for many random events "choosing" the less probable option. If path A is 1 units longer than the best path found so far, and path B is 2 units longer, then twice as many ants would have to randomly choose Path B than A for their pheromone levels to stay the same. Only if B is randomly selected more than twice as much in such succession so that its increased pheromone levels gives it twice the probability of being selected compared to A, only then could the pheromones take over to select a suboptimal path. This issue can be mitigated by decreasing the pheromones that ants spread, requiring more improbable events in a row to hit this critical value. Relevant is also the evaporation rate, with a small evaporation rate, incorrectly enforced paths will stay for longer, increasing the chance of reinforcing the bad path. Following the logic above, we can control and decrease the probability of "accidentally" reinforcing a suboptimal solution by increasing the amount of ants, decreasing their individual pheromone spread, and increasing the fraction of pheromones that are evaporated.

ACO is clearly much more effective than both a greedy best first search (https://en.wikipedia.org/wiki/Best-first_search) and a brute force approach. The greedy best first would be very fast, especially if only run a single time, but has no process to avoid getting stuck in a local minima, which also means it often does not find the optimal solution, depending on the complexity of the problem. Brute force on the other hand will guaranteed find the best solution, although as it has a time complexity of $O(n!)$, this algorithm becomes unfeasibly slow as n increases (n being the amount of nodes in the network). Greedy best first and brute force are on the two extremes on the balance between quality of result and time complexity. ACO has a much better balance, as it quickly eliminates large parts of the solution space while keeping a broad exploration.

Greedy best first search could also get stuck in local minima which are very bad solutions. If ACO gets stuck in local minima, it will with high probability be near optimal. Since the incrementing of the pheromones is proportional to the difference of the solutions and the best solution so far, it is much easier to get out of a local minima which is far away from best so far. Local minima that have a large difference of the objective function compared to the lowest point found so far become more volatile, or unstable. This stays true if the best solution found so far is close to the global minima, which is in general the case for ACO because of how it traverses the solution space. As covered in detail by (Gómez & Barán, n.d.), one of

the reasons for ACOs success in TSP is its ability narrow down the solution space. This can loosely be explained by the fact that near-optimal solutions for the TSP would share many edges, which is in ACOs favor as it's an edge selection based algorithm. Put in more concrete words, it's highly improbable that there's a path which is significantly better than the best path found so far, which does not share any edges. Another way to phrase it is that the solution space has a convex structure at a larger scale.

Comparison, GA and ACO

Comparing a genetic algorithm (GA) and ACO is interesting as when you look at them conceptually, they in general build on similar principles when it comes to TSP (Gómez & Barán, n.d.). A similarity is that both algorithm can be seen as working with some kind of population, which has direct and combined impact on the further state. This is clear in GA, but also in ACO if you consider the edges as individual chromosomes, subject to high elitism. They performing crossover by increasing the probability that edges connected to a "high-pheromone" edge get searched, meaning that two "high-pheromone" edges can effectively merge, by increasing the amount of ants running over their nodes, and probabilistically favoring the edges that connect these edges. Mutation, or exploration in the solution space, is performed by the random selections of each individual ant. These similarities are of course very conceptual, the largest practical difference are how this "crossover" and "mutation" is performed. The crossover in ACO is on a local scale, while GA does crossover between two whole paths. In ACO the crossover does not make large steps from the "parents" in the solution space, instead incremental steps which favor improving objective function, the negative of this is that it's not as exploratory as the GA crossover, which can mix entire paths in different ways. The mutation is also different, in ACO it has a very frequent but small effect, while in GA it's more rare but has more extreme effects (Depends on how it's performed, but a common approach, switching place of two cities, has a large impact on the objective function).

It has been shown that ACO outperforms GA in Path-finding problems which have similarity to TSP. In the test performed by (Binti Sariiff & Buniyamin, 2010), ACO was more than twice as fast in finding the optimized path.

Possible improvements

There are multiple possible improvements for this algorithm. One is simply to run the algorithm multiple times with different hyperparameters, measuring the overall performance and finding the optimal hyperparameters for the specific use case. Another is simply performing the simulation in parallel, conceptually this could be like having multiple colonies of ants which are not affected by each others pheromones, this would decrease the probability of getting stuck in local minima (Manfrin et al., 2006). Another interesting example of an improvement is making the pheromone distribution rank-based, similar to rank-based selection in GAs, we would only allow the top n ants spread pheromones, we could also make this pheromone spread proportional to the ants rank instead of its performance (Dorigo, 1999). More developed versions of ACO, and how they work, is covered by (Gómez & Barán, n.d.).

[#emergentproperties]

This algorithm relies on how the pheromone accumulation on the graph edges is an emergent property of learning between the ants. Every single ant walks pseudorandomly and sprouts pheromones in proportion to the reciprocal of distance traveled, no ant has any understanding of the overall path, they also don't learn anything as they don't adapt their behavior, they only follow the pheromones laid by other equally dumb ants. The incremental effect the pheromone has on the probabilities of following ants decisions lead to a positive feedback loop, reinforcing the edges which often take part in paths which end up short. No ant has any sense of the shortest path, but the pheromone concentration represents the collective learning of many ants and their decisions. This emergent property is exactly what we observe in the animation for the graph, where we can intuitively see how this collective learning adapts, changes its mind, has different certainty, and finally converges upon a reinforcing decision.

PART 2: SIMULATION

The SIR model of the spread of disease is commonly used to help understand how a disease might move through a population. You were introduced to this with the NetLogo agent-based model in NS50 and will review it again in Weeks 7 and 8 of CS51. Check out the class readings to learn about this model.

For this assignment, you will select one disease of your choice to model. Please choose a disease from [THIS LIST](https://docs.google.com/spreadsheets/d/1_p_6W8Zh07XFwuQ3EzkT7PEdbtwTzejHbpKm_wBOLas/edit?usp=sharing) (https://docs.google.com/spreadsheets/d/1_p_6W8Zh07XFwuQ3EzkT7PEdbtwTzejHbpKm_wBOLas/edit?usp=sharing) to investigate. If you would like to select your own disease to model, you may email your professor with the disease and parameter descriptions for approval. You must select an infectious disease (one that is transmitted from person to person through a viral, bacterial, or parasitic agent), not a genetic or environmental disease.

Part 2.1 Numerical Modeling and Simulation

For this part of the assignment, you'll consider the SIR model described by the set of differential equations below, and the numerical simulation in Python via Euler's method.

$$\frac{dS}{dt} = -\frac{b}{N}S(t)I(t)$$

$$\frac{dI}{dt} = \frac{b}{N}S(t)I(t) - kI(t) - mI(t)$$

$$\frac{dR}{dt} = kI(t)$$

$$\frac{dD}{dt} = mI(t)$$

2.1.1 Variables and Parameters (~250 words) [#variables]

This section serves to set up an initial analysis of the SIR model.

1. State the disease you selected to model.
2. Identify the relevant **variables** of the model, fully classify what type of variables they are, and explain what they mean in the context of your model. Next, identify appropriate numerical values and units for the **initial values** of the variables. You're encouraged to use empirical data if possible to justify these values. You may also complete a well-reasoned *#estimation* for any values that are difficult to justify with empirical data. Include APA citations for any external sources used. Note that you can work with population values S , I , R , or proportions, S/N , I/N , R/N , as long as you are consistent.
3. Explain what the relevant **parameters** (b and k) are and what they mean. For your model, identify and justify appropriate numerical values and units for b and k . As above, you may include a well-reasoned estimate using empirical data to support your justification. Further, explain what it would mean for the parameters (b and k) to be smaller or larger. Consider what real-world factors, in the context of the disease you selected, would reduce or increase these parameters.
4. *Optional*: Modify the basic SIR model to add a layer of real-world complexity. A few ideas are listed below. Explain the key features of the extended model, including the modified differential equations and a full description and classification of any new variables and parameters following the steps above.
 - Vaccination
 - Antibiotic use and/or development of antibiotic resistance
 - Variability in population susceptibility (e.g. children and the elderly have different rates of infection compared to young adults).
 - Birth and death rates in the population

I chose to model the disease Ebola.

The variables are **S**, **I**, and **R**, and they represent the amount of people in the population which are **S**usceptible, **I**nfected, and **R**emoved. In the real world these variables will be discrete, as their unit are 'amount of people'. Although in Euler's method we'll use them as continuous functions, eg $S(t)$. We do this because derivation, which Euler's method uses, requires a continuous function. The variable **N**, the size of the population, is discrete, as we don't need to perform derivation on **N**. **t** is the independent variable and represents time, a continuous variable used in a discrete way for Euler's Method, as it relies on an iterative process where we increment **t** with a certain step size.

b and **k** are parameters, which represent the infection and recovery rate respectively. Increasing **k**, would be like giving treatment earlier, as that would decrease the time each infected needs to get recovered. Increasing **b**, for example by not cleaning your hands, would mean that more people get infected per infected person. The unit for **b** is 1/time

Ebola was first reported in Liberia in March 30, 2014. (Nyenswah et al., 2016)

The population at that time was, ~4.36 million (worldbank.org, 2014)

The infection and recovery rates are .8 and .5, the fatality rate being 55-60%. (Pan American Health Organization, n.d.)

$$S_0 = 4\,359\,999$$

$$I_0 = 1$$

$$R_0 = 0$$

$$b = 0.8$$

$$k = 0.5$$

time unit is days

2.1.2 Euler's Method Description (~150 words) [#algorithms]

Explain what it means to solve the SIR differential equations and how Euler's method works as an algorithm to achieve a numerical solution via simulation. In your explanation, identify the inputs, outputs, and steps that the algorithm takes, and consider the role of the step size (h) in the algorithm.

Euler's method solves the SIR differential equations numerically, that means that the output is not a continuous function that is a solution to the differential equation, instead the output is a series of points which estimate the function. The inputs for Euler's method are an initial point, and a differential equation for the derivative. By calculating the derivative at the initial point, we create a new point one step size away with the assumption that the slope stayed the same between these two points. When the step size h is large, this assumption will fail, as the slope will curve between two points, this error will decrease as we decrease h .

2.1.3 Euler's Method Implementation [#algorithms, #dataviz]

Define a function that implements a numerical simulation to derive the implications of your model using Euler's method in Python. Your simulation must generate at least one relevant visualization of the disease dynamics (see required analysis below), including a descriptive figure legend and caption. You may need to adjust the run-time and step size in your simulation to ensure the visualization is maximally informative. Include thorough comments in your code to convey your understanding of the implementation of Euler's method.

Importing Libraries

In [30]:

```
1 import numpy as np
2 import plotly.graph_objects as go
```

Defining Function

```

In [31]: 1 def EulerSIR(infRate,
2         recRate,
3         initialConds,
4         mortalityRate = 0,
5         titleText = "",
6         ss = 1, # step size, I chose 'ss' instead of 'h' for readability
7         tEnd = 100):
8
9     # The amount of days (or other time unit) to simulate is tEnd
10    # Amount of steps to simulate is days/stepsize + 1
11    # (+1 to account for first measurement)
12    stepsAmount = tEnd//ss + 1
13
14    # Create arrays for each population category.
15    # Creating the full array here is faster computationally
16    # than appending values as we calculate them
17
18    # For consistency I usually only use capitalized letters for function names
19    # but I make an exception here considering they represent a function, but discretely
20    S = np.zeros(stepsAmount)
21    I = np.zeros(stepsAmount)
22    R = np.zeros(stepsAmount)
23    D = np.zeros(stepsAmount)
24
25    # Set the initial conditions
26    popSize = sum(initialConds)
27    S[0] = initialConds[0]
28    I[0] = initialConds[1]
29    R[0] = initialConds[2]
30    D[0] = initialConds[3]
31
32    for n in range(stepsAmount - 1):
33        # I use more variables (more storage) than necessary here
34        # because it increases readability
35        newRemoved = ss*recRate*I[n]
36        newInfected = ss*infRate*I[n]*S[n]/popSize
37
38        S[n+1] = S[n] - newInfected
39        I[n+1] = I[n] + newInfected - newRemoved
40        R[n+1] = R[n] + newRemoved * (1 - mortalityRate)
41        D[n+1] = D[n] + newRemoved * mortalityRate

```

```

42
43     GraphSIR(S, I, R, D, titleText)
44
45
46 def GraphSIR(S, I, R, D, titleText):
47     # Define the frames for the animation
48     frames=[dict(data= [dict(type='scatter',
49                             y=S[:k+1]),
50                             dict(type='scatter',
51                                 y=I[:k+1]),
52                                 dict(type='scatter',
53                                     y=R[:k+1]),
54                                     dict(type='scatter',
55                                         y=D[:k+1])),traces=[0,1,2,3],)for k in range(1, len(S)-1)]
56
57     # The different lines to be animated
58     traces = []
59     traces.append(go.Scatter(y=S[:2],
60                             mode='lines',
61                             line=dict(width=1.5),
62                             line_color='green',
63                             name="Susceptibles"))
64     traces.append(go.Scatter(y = I[:2],
65                             mode='lines',
66                             line=dict(width=1.5),
67                             line_color='red',
68                             name="Infectious"))
69     traces.append(go.Scatter(y = R[:2],
70                             mode='lines',
71                             line=dict(width=1.5),
72                             line_color='blue',
73                             name="Recovered"))
74     traces.append(go.Scatter(y = D[:2],
75                             mode='lines',
76                             line=dict(width=1.5),
77                             line_color='black',
78                             name="Deceased"))
79
80     # Layout settings
81     layout = go.Layout(width=700,
82                         height=600,
83                         showlegend=True,

```

```

84     hovermode='x unified',
85     updatemenus=[dict(
86         type='buttons', showactive=False, # Settings for play button
87         y=1.15,
88         x=1.15,
89         xanchor='right',
90         yanchor='top',
91         pad=dict(t=0, r=10),
92         buttons=[dict(label='Play',
93             method='animate',
94             args=[None,
95                 dict(frame=dict(duration=1,
96                     redraw=True),
97                     transition=dict(duration=0),
98                     fromcurrent=True,
99                     mode='immediate')])
100             )]
101         )]
102     )
103
104     # Write titles
105     fig = go.Figure(data=traces, frames=frames, layout=layout)
106     fig.update_xaxes(title_text="<b>Time elapsed</b><br>days")
107     fig.update_yaxes(title_text="<b>Amount of people</b>")
108     fig.update_layout(title_text=f"<b>SIR Model for Ebola</b> in Liberia<br>{titleText}",
109                       yaxis_range=[-2.5*10**5, 4.6*10**6],
110                       xaxis_range=[0, len(S)-1])
111     fig.show()

```

Function Call


```

In [32]: 1 infectionRate = .8
          2 recoveryRate = .5
          3
          4 totalPop = 4.36*10**6
          5 initialInfected = 1
          6
          7 # Order of data is [S, I, R, D]
          8 initialConditions = [totalPop - initialInfected, initialInfected, 0, 0]
          9
          10 mortalityRate = 0
          11
          12 EulerSIR(infectionRate,
          13               recoveryRate,
          14               initialConditions,
          15               mortalityRate,
          16               "Where mortality rate is 0")
          17
          18 mortalityRate = .55
          19
          20 EulerSIR(infectionRate,
          21               recoveryRate,
          22               initialConditions,
          23               mortalityRate,
          24               "Where mortality rate is 55%")
          25
          26 # PRESS PLAY BUTTON TO PLAY ANIMATION

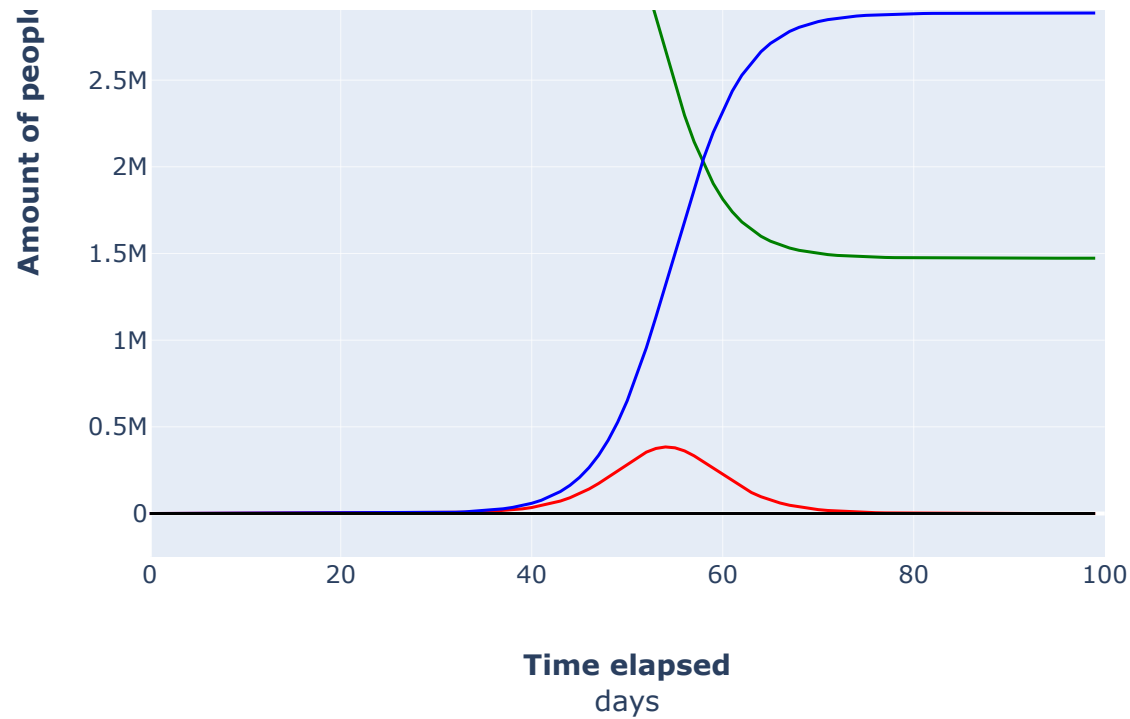
```

SIR Model for Ebola in Liberia

Where mortality rate is 0

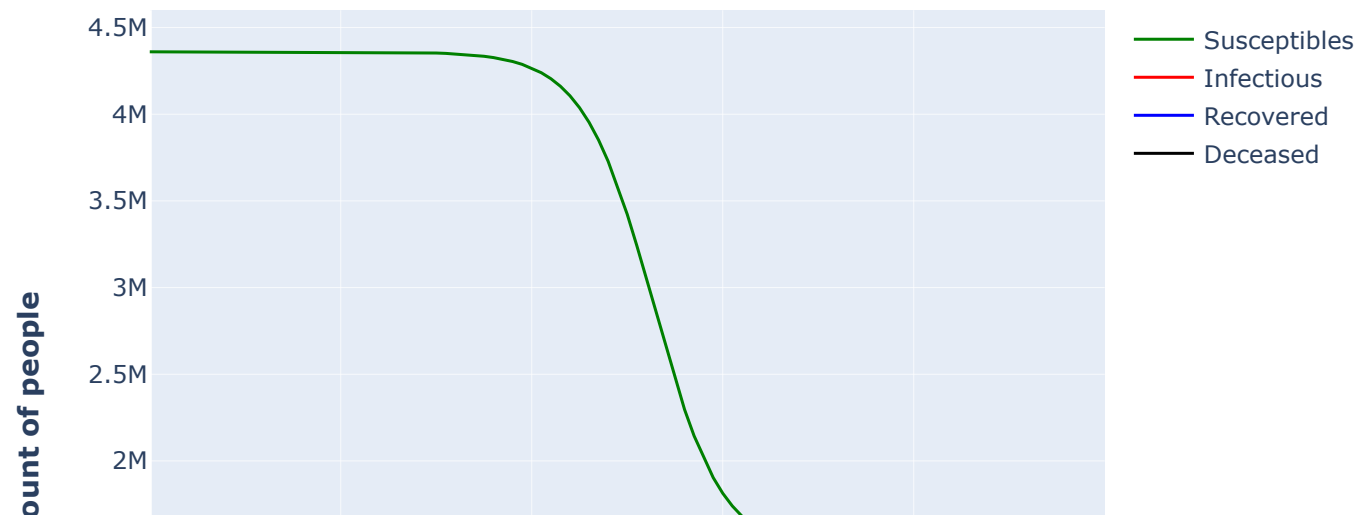
Play

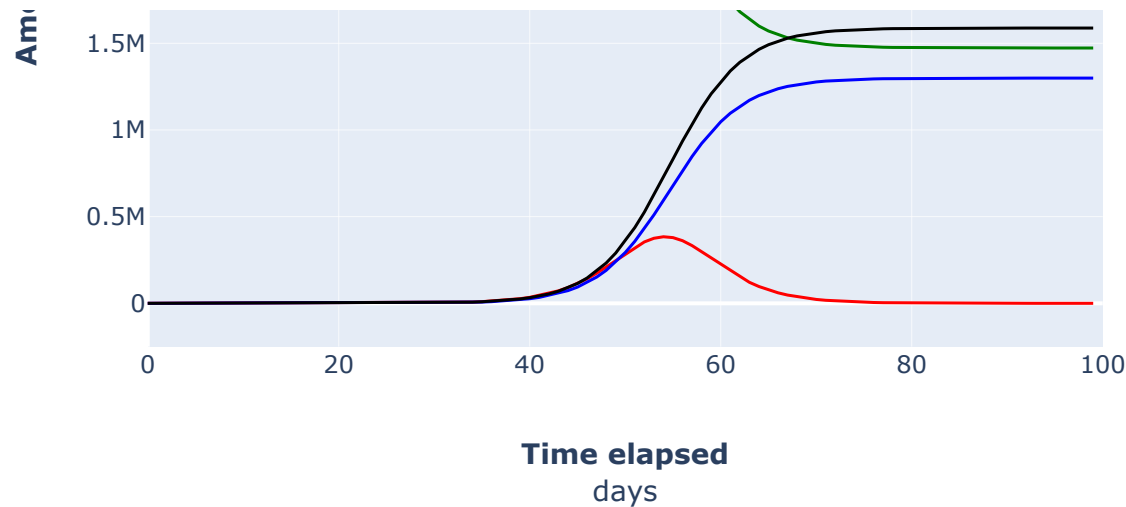




SIR Model for Ebola in Liberia

Where mortality rate is 55%

Play



2.1.4 Results and Interpretation (~250 words) [#modeling, #dataviz]

- Interpret the results of the numerical simulation by making reference the output in the visualization(s). To fully interpret the results, you should re-run the simulation above multiple times with varying parameter inputs (b and k) and observe the behavior of your model. Include at least two additional visualizations here to support your answer. Does the behavior align with what you would expect these adjustments to have in reality (given your answer to 2.1.1.3 above)?
- *Optional*: include at least one multidimensional phase space plot and provide a full interpretation of what it shows.
- Explain how useful this model is by considering the following guiding questions: What insights can be gained? How closely do the results match what you'd expect in reality? What are the most notable assumptions of this model and what impact do they have on its usefulness?

```

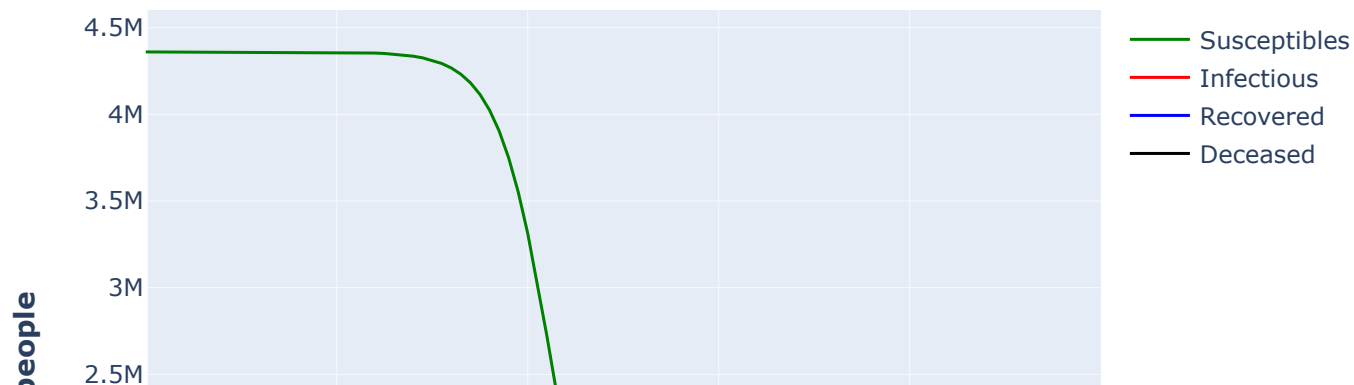
In [33]: 1 totalPop = 4.36*10**6
          2 initialInfected = 1
          3 # Order of data is [S, I, R, D]
          4 initialConditions = [totalPop - initialInfected, initialInfected, 0, 0]
          5
          6 infectionRate = .8
          7 recoveryRate = .4
          8 mortalityRate = .55
          9
          10 EulerSIR(infectionRate,
          11               recoveryRate,
          12               initialConditions,
          13               mortalityRate,
          14               "Where recovery rate is 0.4")
          15
          16 infectionRate = .8
          17 recoveryRate = .6
          18
          19 EulerSIR(infectionRate,
          20               recoveryRate,
          21               initialConditions,
          22               mortalityRate,
          23               "Where recovery rate is 0.6")

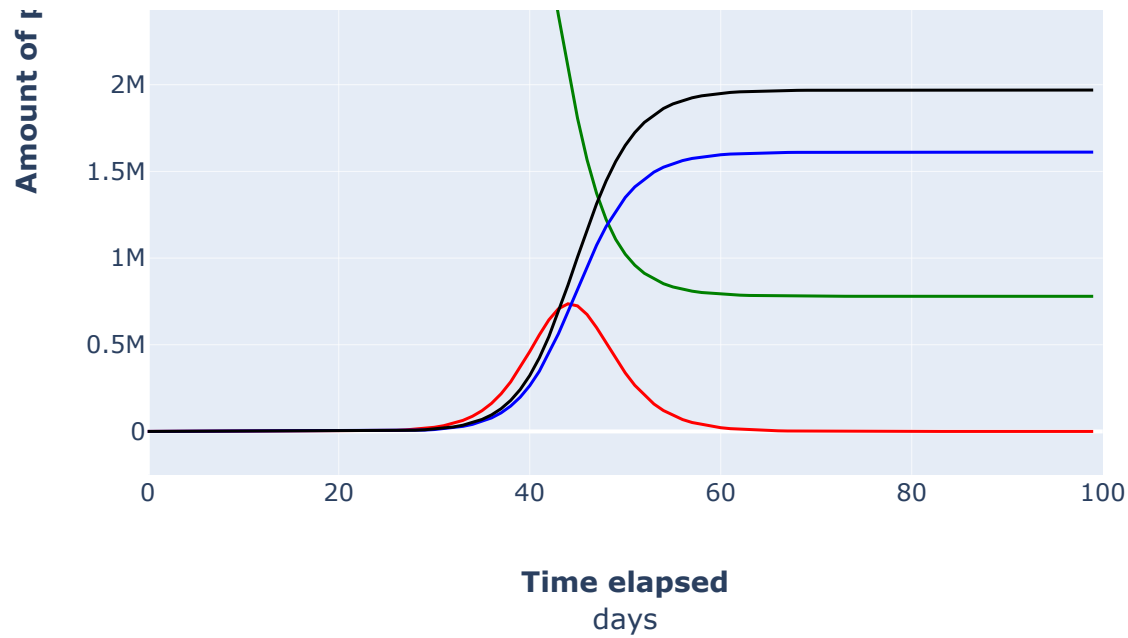
```

SIR Model for Ebola in Liberia

Where recovery rate is 0.4

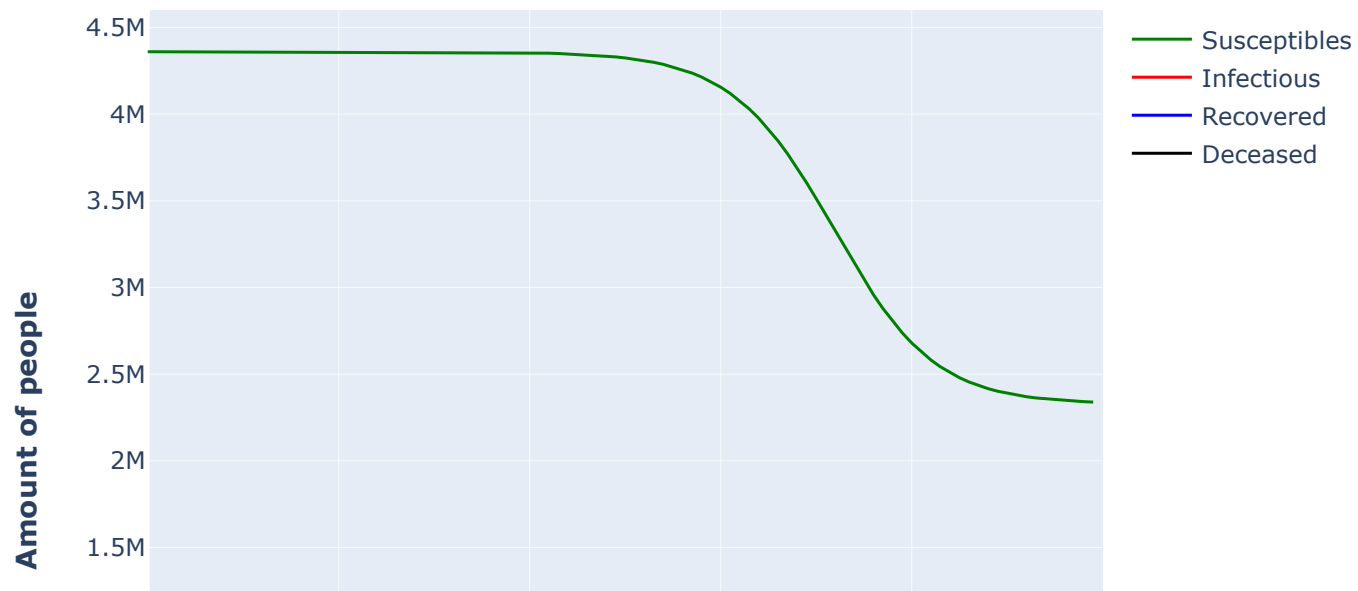
Play

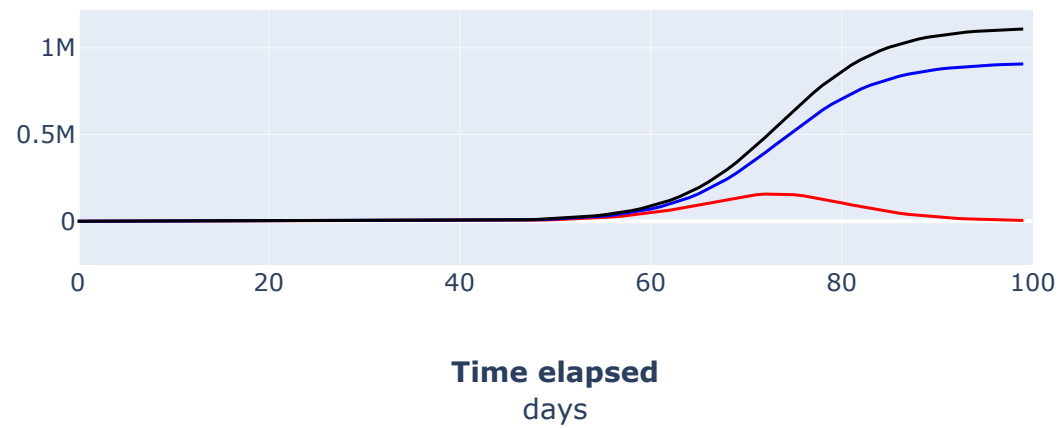




SIR Model for Ebola in Liberia

Where recovery rate is 0.6

[Play](#)



```

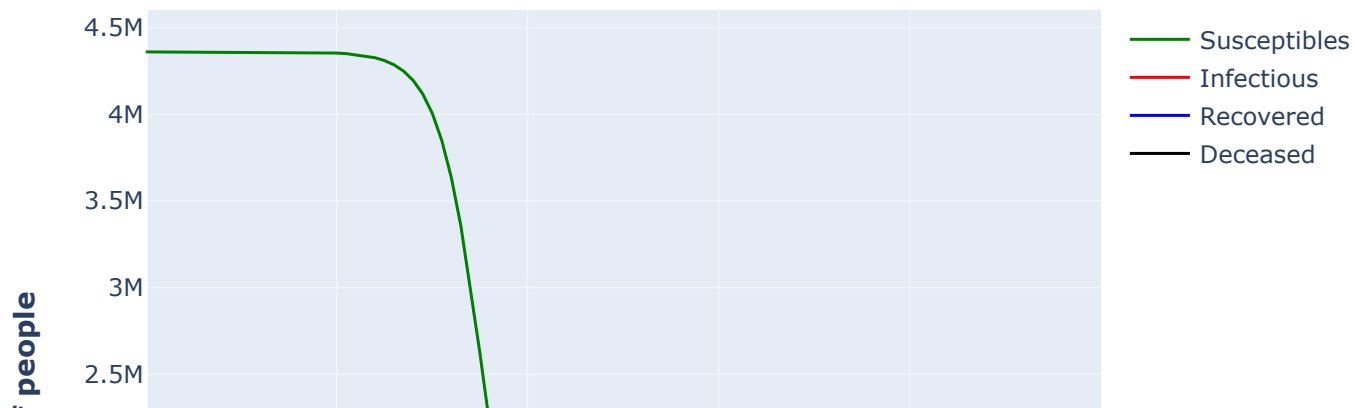
In [34]: 1 totalPop = 4.36*10**6
          2 initialInfected = 1
          3 # Order of data is [S, I, R, D]
          4 initialConditions = [totalPop - initialInfected, initialInfected, 0, 0]
          5
          6 infectionRate = 1
          7 recoveryRate = .5
          8 mortalityRate = .55
          9
          10 EulerSIR(infectionRate,
          11               recoveryRate,
          12               initialConditions,
          13               mortalityRate,
          14               "Where infection rate is 1")
          15
          16 infectionRate = .7
          17 recoveryRate = .5
          18
          19 EulerSIR(infectionRate,
          20               recoveryRate,
          21               initialConditions,
          22               mortalityRate,
          23               "Where infection rate is 0.7")

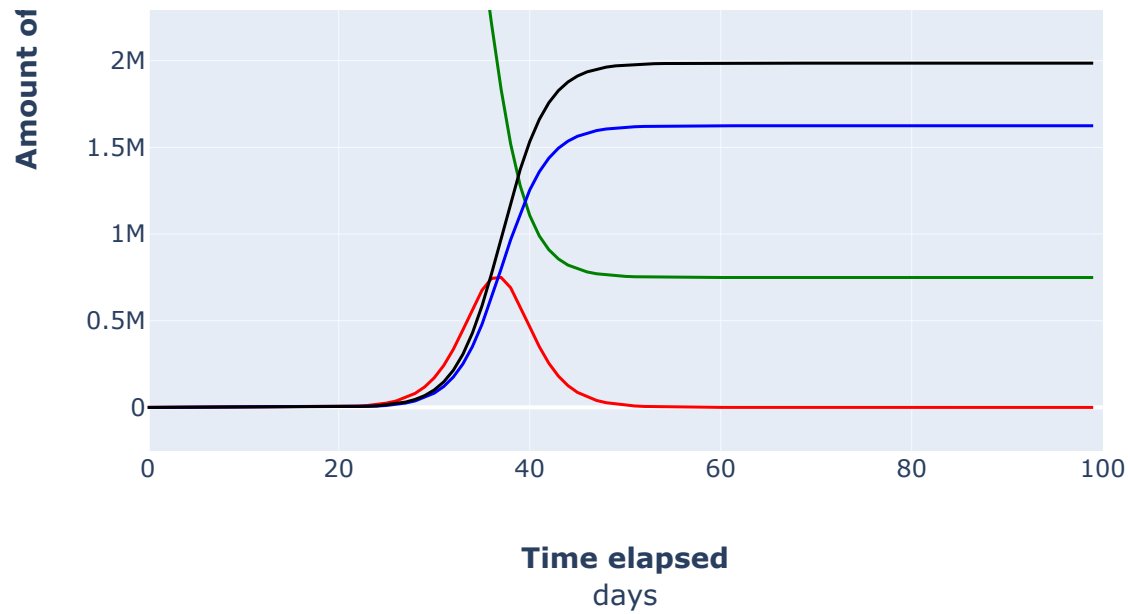
```

SIR Model for Ebola in Liberia

Where infection rate is 1

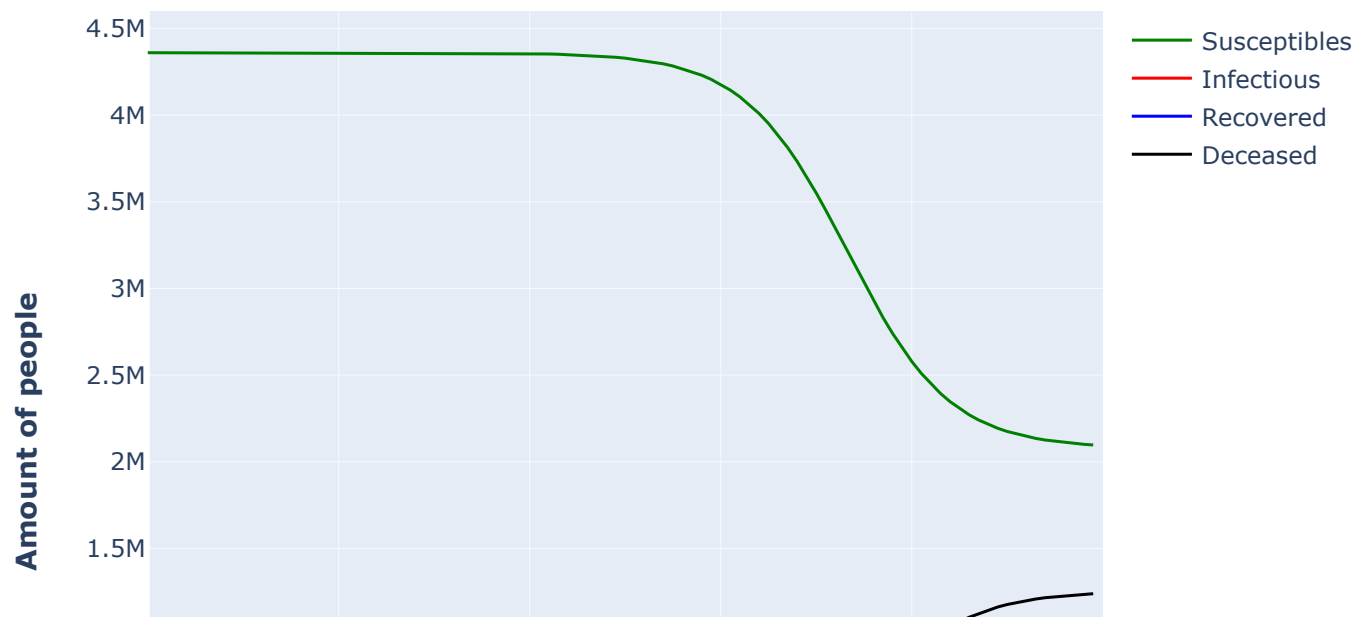
Play

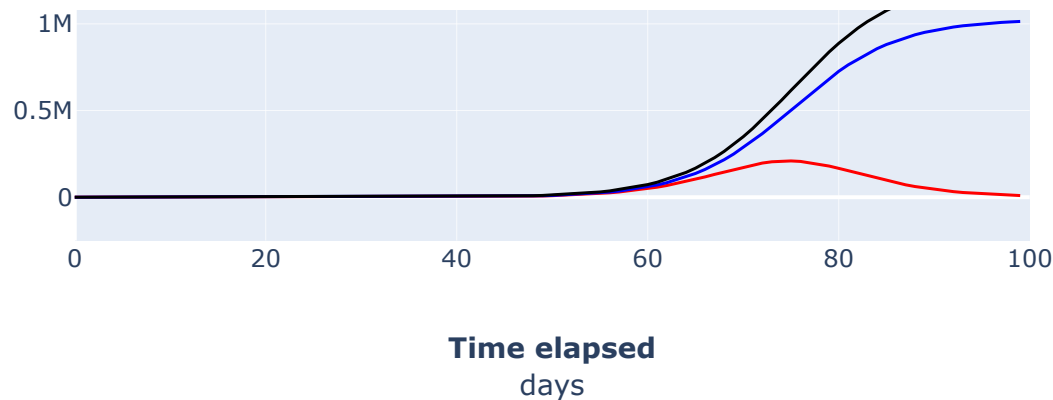




SIR Model for Ebola in Liberia

Where infection rate is 0.7

[Play](#)



In the output we can see the development of the functions **S**, **I**, and **R**, over time. The behavior we see is a single hump of infections, which represents a temporary outbreak, leading to a burst of susceptible people moving to 'removed'. From this graph we can get valuable insight for the real world, how long the outbreak will be, the peak amount of infected people per day, and how many people will end up dying from the outbreak. We can also see what kind of impact real word interventions would have, such as cleaning your hands or improving healthcare, which would be associated with decreasing the infection and recovery rate respectively. We can see from the graphs above that it's the relationship between infection and recovery rate that impacts the length and timing of the outbreak. When the infection rate is about twice the recovery rate, we get the early and high peak, in the other two graphs where infection rate is approximately 1.5 times the recovery rate, we get the slower outbreak that starts around day 60.

Part 2.2 Agent-Based Modeling and Simulation (*OPTIONAL*)

This part of the assignment is optional and will only be scored if completed effectively (score of 4 or 5). It is a valuable chance to compare the simulation above with the agent-based simulation implemented in [NetLogo](http://ccl.northwestern.edu/netlogo/models/Virus) (<http://ccl.northwestern.edu/netlogo/models/Virus>).

Note about parameters: this model uses similar variables and parameters as the one above, but the parameters are not defined identically. In particular, the "Infectiousness" parameter in NetLogo is analogous to, but not equivalent to the infection rate in the SIR model. Thus, they should not be set to the same value in both of your simulations. The infection rate in the SIR model already incorporates the interaction rate of individuals, while the NetLogo simulation sets that rate separately. In other words, the "infectiousness" parameter in NetLogo only dictates the probability of infecting someone if they come close enough, but does not take into account how frequent those interactions occur. Be sure to investigate the meaning of the other parameters as well so that you understand how to set them appropriately.

2.2.1 *Optional*: Simulation Comparison (~250 words) [#modeling]

After fully exploring the NetLogo model and running multiple simulations, summarize how it compares to your Python SIR simulation above. Aim to identify the main similarities, the major differences, and at least advantage for each one. Comment on which you believe to be a more realistic representation of nature, justifying your reasoning.

Answer 2.2.1 here

2.2.2 *Optional challenge*: Your own agent-based simulation [#algorithms, #modeling]:

Create your own agent-based simulation of the disease dynamics for your chosen disease in Python. You may add in real-world complexities as desired (vaccination, antibiotic use and/or development of antibiotic resistance, variability in population susceptibility). Your work needs to be explained in sufficient detail, including citations to any external sources consulted, in order to receive credit.

- One option: a tree graph can be useful in modeling person-to-person interactions.
- Another option: turtles.

```
In [ ]: 1 # Run this to install pygame
        2 import sys
        3 !{sys.executable} -m pip install pygame
```

Import Libraries

```
In [6]: 1 import plotly.express as px
        2 import plotly.graph_objects as go
        3 from IPython.display import clear_output
        4 import numpy as np
        5 import pygame
        6 import math
        7 import random as rnd
        8 import time
```

Define Functions

```

In [19]: 1 # Update the state of dot to become newState
2 def UpdateState(dot, newState):
3     currentDistribution[state[dot]] -= 1 # Decrement the category dot is leaving
4     state[dot] = newState # Change the state of dot
5     currentDistribution[newState] += 1 # Increment the category dot entered
6
7 # Draw the population distribution
8 def UpdateGraph():
9     # Add the current distribution as a new datapoint
10    for i in range(len(statusHistory)):
11        statusHistory[i].append(currentDistribution[i])
12
13    # Labeling and coloring
14    titles = ["Susceptible", "Infectious", "Recovered", "Dead", "Quarantined"]
15    colors = ["blue", "red", "green", "black", "purple"]
16
17    fig = go.Figure()
18
19    # Draw a line for each category
20    for i in range(len(statusHistory)):
21        fig.add_trace(go.Scatter(y=statusHistory[i], mode='lines',
22                                name=titles[i],
23                                line_color=colors[i]))
24
25    fig.update_layout(title_text='''<b>Amount of people per each SIRDQ category in the population</b><br>
26                        over time''')
27    fig.update_yaxes(title_text="<b>Amount of people</b>")
28    fig.update_xaxes(title_text="<b>Elapsed time</b> in weeks")
29    # Clear output before drawing new figure
30    clear_output(wait=True)
31    fig.show()

```

Simulation

In [23]:

```
1  # Run pygame
2  pygame.init()
3
4  # Set size for screen
5  width, height = 800, 600
6  size = 1000, 800
7  screen = pygame.display.set_mode(size)
8  # Amount of seconds between each graph update
9  graphUpdateFrequency = 1
10 # frames per seconds
11 fps = 100
12
13 # Specify populationsize
14 numDots = 300
15
16 # All metadata for each dot
17 dotPos = [] # Position of each dot
18 targetDir = [] # The target direction vector to travel in
19 currentDir = [] # The current direction vector
20 dotTimers = [] # An internal timer, purpose depends on state
21 passedQTime = [] # COMMENT ON THIS
22 # The state of the dots represented as a number
23 # S = 0, I = 1, R = 2, D = 3, Q = 4
24 # this number is equal to the index of which the categories
25 # are stored in 'currentDistribution' [S, I, R, D, Q]
26 state = []
27
28 # Initialize the population
29 for i in range(numDots):
30
31     # randomly position dots
32     dotPos.append(np.array([rnd.random()*width, rnd.random()*height]))
33
34     # Set a random target direction vector
35     angle = rnd.random()*math.pi*2
36     targetDir.append([math.cos(angle), math.sin(angle)])
37     currentDir.append([0, 0])
38
39     # Set all to susceptible
40     state.append(0)
41     dotTimers.append(0)
```

```
42     passedQTime.append(False)
43
44     # Initial condition for population
45     statusHistory = [[numDots], [0], [0], [0], [0]]
46     currentDistribution = [numDots, 0, 0, 0, 0]
47     # Make one random individual infected
48     UpdateState(rnd.randint(0, len(state) - 1), 1)
49
50     # White background color
51     backgroundClr=[255,255,255]
52     # The color for the dots depending on their state
53     # Order is [S, I, R, D, Q]
54     dotColors = [[0,0,255], [255,0,0], [0,255,0], [0, 0, 0], [191, 64, 191]]
55
56     # The size of the dots
57     dotSize = 5
58     # The average amount of seconds each dot takes before they select a new target direction to travel in
59     avgTimeTurn = 1
60     # How quickly/sharply the dots adapt to the new direction vector
61     turningSpeed = 1
62     # The maximum speed of the dots
63     maxSpeed = 2
64
65     # The distance within infected people can spread the disease
66     infectDist = 20
67     # Time in seconds within proximity that counts as one interaction
68     suffInteractionTime = .5
69     # Probability of one interaction spreading the disease
70     infProb = .5
71     # The simulation does not wait for an interaction to occur,
72     # instead the probability of an infection happening each iteration is infProb/(fps*suffInteractionTime).
73     # This effectively means that a susceptible dot will on average become infected after
74     # spending 'infProb/suffInteractionTime' seconds within the 'infectDist' if an infected dot
75
76     # Set amount of seconds each dot spends as infected before becoming recovered or dead
77     recoveryTime = 6
78     # Fraction of how many of infected die
79     mortalityRate = .2
80     # The fraction of recovered dots that will become susceptible
81     probLoseImmunity = .05
82     # The set amount of seconds spent as recovered, if dot will become susceptible again
83     timeFreqLoseImmunity = 20
```

```
84
85 # What fraction of infected we can detect the disease, and then quarantine
86 # Set this to zero to test without quarantining
87 quarantineRate = .9
88 # How long time it takes to quarantine after start of disease
89 quarantineDelay = 3
90 # How many people the quarantine facility can take
91 quarantineCapacity = 50
92 # I have a smaller mortality rate for people quarantining, which represents increased quality of treatment
93 qMortalityRate = .05
94
95 # iteration counter
96 iteration = 0
97
98 running = True
99 while running:
100
101     # Terminate pygame if exit button is pressed
102     for event in pygame.event.get():
103         if event.type == pygame.QUIT:
104             running = False
105
106     # Terminate if disease disappeared
107     if currentDistribution[1] == 0:
108         running = False
109
110     # fill background
111     screen.fill(backgroundClr)
112
113     # For each dot
114     for i in range(len(dotPos)):
115
116         # Draw dots with updated positions and colors
117         pygame.draw.circle(screen, dotColors[state[i]], dotPos[i], dotSize)
118
119         # If dot is infected
120         if state[i] == 1:
121             # Draw a radius circle, visualizing 'infectDist'
122             pygame.draw.circle(screen, dotColors[state[i]], dotPos[i], infectDist, 1)
123
124             # Increment timer
125             dotTimers[i] += 1/fps
```

```

126
127     # If 'quarantineDelay' time has passed, and is first time we check for quarantining
128     if dotTimers[i] > quarantineDelay and not passedQTime[i]:
129         passedQTime[i] = True
130         # If disease is detectable, so we can quarantine, and if there's capacity
131         if rnd.random() < quarantineRate and currentDistribution[4] < quarantineCapacity:
132             # Convert to quarantined
133             UpdateState(i, 4)
134             dotTimers[i] = 0
135             # Move to quarantine box
136             dotPos[i] = [800 + rnd.random() * 100 - 50, 700 + rnd.random() * 100 - 50]
137
138     # If 'recoveryTime' spent as infected
139     if dotTimers[i] > recoveryTime:
140         # Reset timer
141         dotTimers[i] = 0
142
143         # move dot to deceased according to 'mortalityRate'
144         if rnd.random() < mortalityRate:
145             UpdateState(i, 3)
146         # Else, become recovered
147         else:
148             UpdateState(i, 2)
149
150     # If quarantined
151     if state[i] == 4:
152         # When 'recoveryTime' time has passed
153         dotTimers[i] += 1/fps
154         if dotTimers[i] > recoveryTime:
155             # move dot to deceased according to quarantine 'qMortalityRate'
156             if rnd.random() < qMortalityRate:
157                 UpdateState(i, 3)
158             # Else, become recovered
159             else:
160                 UpdateState(i, 2)
161
162     # If is recovered and timer has not been paused
163     if state[i] == 2 and dotTimers[i] != -1:
164         # Update timer
165         dotTimers[i] += 1/fps
166
167     # If 'timeFreqLoseImmunity' has been spent as recovered

```

```

168     if dotTimers[i] > timeFreqLoseImmunity:
169         # Make dot susceptible according to 'probLoseImmunity'
170         if rnd.random() < probLoseImmunity:
171             UpdateState(i, 0)
172         # Else, pause timer
173         # I'm not increasing the probability of becoming immune over time,
174         # instead I just calculate this probability once
175         else:
176             dotTimers[i] = -1
177
178     # If dot is not dead or quarantined
179     if state[i] != 3 and state[i] != 4:
180         # Update positions
181         # travel in currentDir with a speed of maxSpeed
182         # magnitude of currentDir can be < 1 when dot is still turning from last direction,
183         # Therefore maxSpeed is speed only when currentDir == targetDir.
184         # I do modulus of the screen dimensions so that the dots can walk out on the sides
185         # and appear on the opposite side. Making the traverse, topologically speaking
186         dotPos[i][0] = (dotPos[i][0] + maxSpeed*currentDir[i][0]) % width
187         dotPos[i][1] = (dotPos[i][1] + maxSpeed*currentDir[i][1]) % height
188
189     # Spread infection
190     if state[i] == 1:
191         for ind in range(len(dotPos)):
192             if state[ind] == 0: # This does not take edge teleportation into account
193                 # If distance is within infection distance
194                 if math.sqrt((dotPos[i][0]-dotPos[ind][0])**2 + (dotPos[i][1]-dotPos[ind][1])**2) < infectDist:
195                     # Calculate probability of infection
196                     if rnd.random() < infProb/(fps*suffInteractionTime):
197                         # Make dot infected
198                         UpdateState(ind, 1)
199                         passedQTime[ind] = False
200
201     # Set new target direction vector, on average each 'avgTimeTurn' seconds
202     if rnd.random() < 1/(fps*avgTimeTurn):
203         # Randomize angle
204         angle = rnd.random()*math.pi*2
205
206         # Create normalized direction vector based on angle
207         targetDir[i][0] = math.cos(angle)
208         targetDir[i][1] = math.sin(angle)
209

```

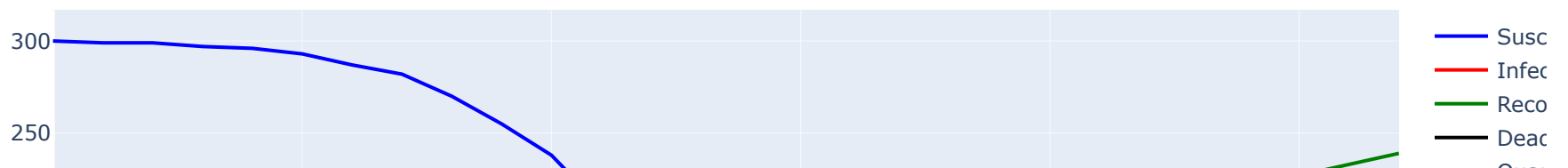


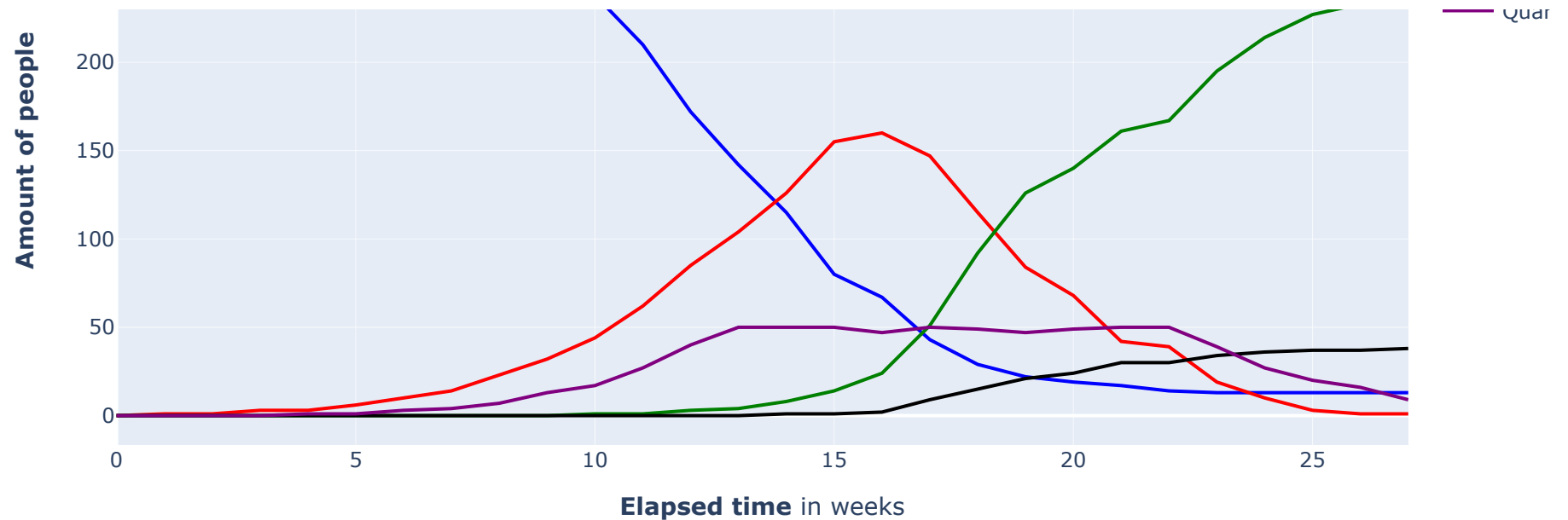
```

210     # Update current direction to move smoothly toward target direction
211     # If x component of current direction is not equal target x component
212     if currentDir[i][0] != targetDir[i][0]:
213         # Move closer to the target value,
214         # change in x component is proportional to 'turningSpeed' and the difference between current and target.
215         # We get smoother turning by having turning be proportional to the difference rather than changing the
216         # direction with a set amount each iteration.
217         currentDir[i][0] += (turningSpeed/fps)*(targetDir[i][0] - currentDir[i][0])
218     if currentDir[i][1] != targetDir[i][1]:
219         currentDir[i][1] += (turningSpeed/fps)*(targetDir[i][1] - currentDir[i][1])
220
221     # Draw quarantine zone
222     pygame.draw.rect(screen,[0, 0, 0],(740, 640, 120, 120), 2)
223
224     # Update the display
225     pygame.display.update()
226
227     # Increment the iteration counter
228     iteration += 1
229
230     # Each 'graphUpdateFrequency' seconds
231     if iteration % (graphUpdateFrequency*fps) == 0:
232         # Update linegraph
233         UpdateGraph()
234
235     # Sleep 1/fps seconds
236     time.sleep(1/fps)
237
238 # Quit pygame if loop is exited
239 pygame.quit()

```

Amount of people per each SIRDQ category in the population over time





Notes

When running the simulation above a pygame window should appear, displaying the simulation. The simulation is not adapted to the disease Ebola, as I don't have sufficient data with good `#sourcequality` to support decisions about hyperparameters, this is instead a more abstract representation of a possible disease

Discussion

Using this agent-based simulation we can include stochastic elements to better represent the random events of the real world, this was not taken into account in the deterministic SIR model which used Euler's method to get a numerical solution. This model has many more hyperparameters, which have an intuitive interpretation for the real world, when running this simulation we can gain understanding about what kind of effect these hyperparameters have on relevant variables such as death cases or peak infectious cases in a week. One valuable lesson we can see here is the impact of quarantining on the outbreak, and how the constraint of a maximum capacity for a quarantine facility can lead to an increase in cases.

REFLECTION

In less than 100 words, explain how this unit has enhanced your view of the power of modeling, algorithms, and simulations to describe the natural world around you.

This unit has increased my understanding of how to model real world scenarios, performing optimization algorithms and model the systems to inform predictions and decision making. It also helped me how to model real world complex systems in the form of differential equations, and how to calculate numerical solutions when analytical ones are unfeasible. I also gained understanding the advantages and limitations of these different algorithms and models. Additionally how agent based models to simulate and observe relevant emergent properties to gain understanding on the dynamics of the system.

You're done!

You must upload TWO files:

1. A **PDF** of your entire assignment. This is to be submitted as a separate file, NOT simply inside the zipped folder. Email attachments will not be accepted. We encourage students to follow the tips available in [this guide](https://docs.google.com/document/d/1gRMol9Ebbvyu1mvEKzma92o_N7ZbNXsPIb1QdQV0TeE/edit?usp=sharing) (https://docs.google.com/document/d/1gRMol9Ebbvyu1mvEKzma92o_N7ZbNXsPIb1QdQV0TeE/edit?usp=sharing), especially the best practices listed at the end.
2. A **zipped folder** containing the .ipynb file and any other relevant files for running the notebook.

Bibliography

Binti Sariff, N., & Buniyamin, N. (2010). GENETIC ALGORITHM VERSUS ANT COLONY OPTIMIZATION ALGORITHM - Comparison of Performances in Robot Path Planning Application. Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics. <https://doi.org/10.5220/0002892901250132>

Dorigo, M. (1999, April). (PDF) ACO Algorithms for the Traveling Salesman Problem. ResearchGate. https://www.researchgate.net/publication/2771967_ACO_Algorithms_for_the_Traveling_Salesman_Problem

Gómez, O., & Barán, B. (n.d.). Relationship between Genetic Algorithms and Ant Colony Optimization Algorithms (p. Section 7.1). Retrieved March 7, 2022, from https://www.cnc.una.py/publicaciones/1_86.pdf

Kumar, R., & Dey, S. (2015). SIR Model for Ebola Outbreak in Liberia. International Journal of Mathematics Trends and Technology, 28(1), 28–30. <https://doi.org/10.14445/22315373/ijmtt-v28p506>

Manfrin, M., Birattari, M., Stützle, T., & Dorigo, M. (2006). Parallel Ant Colony Optimization for the Traveling Salesman Problem. Ant Colony Optimization and Swarm Intelligence, 224–234. https://doi.org/10.1007/11839088_20

Nyenswah, T. G., Kateh, F., Bawo, L., Massaquoi, M., Gbanyan, M., Fallah, M., Nagbe, T. K., Karsor, K. K., Wessah, C. S., Sieh, S., Gasasira, A., Graaff, P., Hensley, L., Rosling, H., Lo, T., Pillai, S. K., Gupta, N., Montgomery, J. M., Ransom, R. L., & Williams, D. (2016). Ebola and Its Control in Liberia, 2014–2015. Emerging Infectious Diseases, 22(2), 169–177. <https://doi.org/10.3201/eid2202.151456>

Pan American Health Organization. (n.d.). Ebola Virus Disease - PAHO/WHO | Pan American Health Organization. [www.paho.org. https://www.paho.org/en/topics/ebola-virus-disease#:~:text=The%20virus%20is%20transmitted%20to](https://www.paho.org/en/topics/ebola-virus-disease#:~:text=The%20virus%20is%20transmitted%20to)

worldbank.org. (2014). Population, total - Liberia | Data. [Data.worldbank.org. https://data.worldbank.org/indicator/SP.POP.TOTL?locations=LR](https://data.worldbank.org/indicator/SP.POP.TOTL?locations=LR)

[#sourcequality]

I use multiple scientific articles which are directly relevant to the context. The sources are of high quality as the authors are of appropriate academic authority, for example Marco Dorigo which has over 90 000 citations. I use multiple sources which can confirm each other, strengthening the reliability.