INTRO TO AI

7.16.21

A2: Solving TSP with Genetic Algorithms

Bryan Perdomo

Dr. Marques Oge

Intro to AI

A2: SOLVING TSP WITH GENETIC ALGORITHMS

CREDITS AND ACKNOWLEDGEMENTS

This implementation of the solution to the Traveling Salesman Problem using Genetic Algorithms draws heavy inspiration from the 'Solving TSP with GA's in Python' Article published by Eric Stoltz provided in the assignment specifications. I cannot stress enough how invaluable this particular resource has been in facilitating my learning of how genetic algorithms work, and how to use them in real world applications. All credit goes to Eric Stoltz, his work, other resources, and research I conducted can be found at the end of this report.

PROJECT NOTES

Below, you'll find the report questions and answers from the assignment specification.

How were the cities and distances represented (as a data structure)?

The cities were represented as a pair of x and y coordinates which correspond to a point in space. A list containing tuples storing these coordinates was the data structure used. Each city in a route was instantiated as a City object, inherently having and x and y coordinate. The distances then, were calculated using the Pythagorean theorem between 2 sets of city coordinates. The route list was looped through, and each city's front, and back city distance were calculated.

How did you encode the solution space?

This implementation encodes the solution space as a permutation representation. The traditional Knapsack problem discussed in the textbook can be encoded using a simple binary representation. Since the TSP

problem has some special considerations, it makes more sense to use a permutation-based representation, since the solution is based on an order of elements. The solution of the TSP in of itself is in fact, an ordering/ permutation of every city that needs to be visited.

How did you handle the creation of the initial population?

Inbuilt functions from the numpy module were leveraged to create the initial population. The implementation involved two functions, the first was used to create an individual(route) by using the sample() function from the numpy.random module to return a random city(pair of x-y coordinates), given an iterable object and size. The second function expands the first by looping through a range(the desired size of your population) and calling the first function, which creates the initial population of cities.

How did you compute the fitness score?

A dictionary is created to store each route in a population and its respective fitness score. We iterate through each route in a population (or list of cities/ xy pairs) and determine how fit this particular route is. In order to actually determine each routes specific fitness score, we iterate through each city(xy-coordinate) in the route, keep track and connect each city, calculate the distance between every city in sequential order, and sum up each distance into a total distance for this route. We make sure to place check to make sure that each city is visited only once, and that the last city is connected to the first so that we start and end at the same xy coordinate. From there, we simply take the inverse of the total distance and return this value as the fitness score. This score is then stored in our dictionary, and the keys are the index positions(route 1, route 2, etc). The result is a dictionary containing a population with its 100 routes and fitness scores.

Which parent selection strategy did you use? Why?

This implementation uses roulette wheel selection to select parents. Speaking candidly, compared to implementing the other selection strategies like rank and tournament selection, roulette wheel proved to

be easier and more straightforward. Something to note is that traditionally, larger fitness scores result in larger slices of the wheel, which correspond to a higher probability of selection, and that is the case here, However, since the TSP deals with distances, this implementation computes the fitness score as the inverse of a routes total distance.

Which crossover strategy(ies) did you try? Which one worked out best?

I personally did not experiment with various crossover strategies, and instead utilized the provided implementation's special breeding function: "Ordered Crossover (OX1) The main motivation for the utilization for this crossover strategy is because of the special constraint inherent in the Traveling salesman problem. Since the problem statement requires that all cities are visited exactly once, the order crossover strategy takes this into account by giving offspring a random subset of one parent and filling in the rest of the offspring with the unused subset from the 2nd parent.

Which mutation strategy(ies) did you try? Which one worked out best??

As previously mentioned, the TSP includes some constraints that beckon a special mutation strategy, such as each city needing to be visited exactly once and not disposing of cities. The one used in this implementation is order/swap mutation. This involves selecting two positions on a genes at random and swapping their values, ensuring all items remain in the chromosome while introducing diversity.

Which strategy did you use for populating the next generation? Why?

To populate the next generation, this implementation first determines fitness by ranking routes, selecting the best routes also using elitism, extracting the routes determined to be most fit, creating offspring via ordered crossover, mutating this population to introduce diversity, and repeating this for several generations.

Which stopping condition did you use? Why?

The stopping condition that was used in this implementation was to hard-code a user input variable that stores the desired number of generations to be created. Additionally, the function that creates new generations takes this variable as a parameter. This effectively serves as the stopping criteria, so for example if 'generations' = 100, then the genetic algorithm will continue to run until 100 generations have been run through.

What other parameters, design choices, initialization and configuration steps are relevant to your design and implementation?

One interesting parameter to note is the use of Elitism with regards to the parent selection strategy. Quoting Baluja et al, "A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection* and guarantees that the solution quality obtained by the GA will not decrease from one generation to the next." Elitism is a way to ensure the fittest, best performing individuals automatically go on to the next generation, and in theory yields more optimal solutions.

Which (simple) experiments have you run to observe the impact of different design decisions and parameter values?

Quite frankly, I didn't run any complicated or involved additional experiments. I did however spend some time playing around with the various parameters that influence the genetic algorithm. For example, by simply changing the number of generations that the algorithm runs through I found that this directly impacts how optimal a solution is yielded. As you would expect, the greater the number of generations, the shorter the distance (so the better the solution). What's Interesting to note however, is it seems that after a certain number of generations, the efficacy and performance gains tend to starkly drop off. Using rough values, and all other parameters being equal, the initial solution for 25 random cities

produces a value of around 1800 - 2400 distance. After 100 generations are run through the genetic algorithm, a solution of approx. 1000 - 1200 distance was observed. One would think that increasing the number of generations would lead to exponentially better results. However, after increasing the number of generations used to 500 (5x), the solution produced is approx. 1000-1100. While not very scientific, this does seem to suggest that the vast majority of performance gains are seen within the creation of the first 5-100 generations, and any additional generations after that yield severely diminished benefits. Screenshots of these results have been included in the report.

Resources and Research

Eric Stoltz Article:

https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35

Stack Overflow:

https://stackoverflow.com/questions/23183862/what-is-the-difference-between-roulette-wheel-selection-rank-selection-and-tour

Parent selection strategy comparison:

https://www.idjsr.com/uploads/68/3180_pdf.pdf

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.3747&rep=rep1&type=pdf

Misc:

https://en.wikipedia.org/wiki/Genetic_algorithm

Baluja, Shumeet; Caruana, Rich (1995). Removing the genetics from the standard genetic algorithm (PDF). ICML.

```
18  #Running genetic Algorithm
19  geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=1)
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/Activate.ps1"
(env) PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/python.exe" "c:/Users/
initial distance: 2222.262076990748
Final distance: 2222.262076990748
(env) PS C:\Users\bbper\Intro to AI\A2>
```

```
18  #Running genetic Algorithm
19  geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=25)
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/Activate.ps1"
(env) PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/python.exe" "c:/Users/b
initial distance: 2398.897848620875
Final distance: 1481.6863515191105
(env) PS C:\Users\bbper\Intro to AI\A2>
```

```
18  #Running genetic Algorithm
19  geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=50)
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/Activate.ps1"
(env) PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/python.exe" "c:/Users/
initial distance: 2140.7316202769994
Final distance: 1170.6757330766052
(env) PS C:\Users\bbper\Intro to AI\A2>
```

```
18  #Running genetic Algorithm
19  geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=75)
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/Activate.ps1"
(env) PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/python.exe" "c:/Users/
initial distance: 2207.5179661453926
Final distance: 1122.1858330148611
(env) PS C:\Users\bbper\Intro to AI\A2>
```

```
18  #Running genetic Algorithm
19  geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=150)
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/Activate.ps1"
(env) PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/python.exe" "c:/Users/b
initial distance: 2048.7179820358797
Final distance: 950.3606345193763
(env) PS C:\Users\bbper\Intro to AI\A2>
```

```
19  geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=500)
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/Activate.ps1"
(env) PS C:\Users\bbper\Intro to AI\A2> & "c:/Users/bbper/Intro to AI/A2/env/Scripts/python.exe" "c:/Users/b
initial distance: 1962.1574504081384
Final distance: 888.5390970563625
(env) PS C:\Users\bbper\Intro to AI\A2>
```