# Assignment 1

**Name: Gu Zhexuan Matriculation number: 22054513g**

**Course: Artificial Intelligence Concepts Lecturer: Professor TAN**

**Outline**

1. **Introduction to GA**
2. **Using GA to tackle simple TSP (problem 1)**
3. **Solution to Dynamic TSP (problem 2)**
4. **Introduction to K-means**
5. **Combine GA and K-means to address large-scale TSP (problem 3)**
6. **Brief introduction to multi-object optimization**
7. **Using weighted-GA to solve problem 4**
8. **Introduction to NSGA-II and make comparison with weighted-GA (problem 4)**
9. **Using NSGA-II to solve multi-objective optimization (problem 5)**
10. **Illustration to python code file structure and function of each file**

# Introduction to GA

GA, which stands for “genetic algorithm”, is a search-based algorithm that is proved to be extremely useful in AI optimization Problem.

There’re basically three operators in GA, they’re **selection, crossover, and mutation**. Since it contains “genetic”, it follows what Darwin has said “survival of the fittest”. In a specific problem, it means that the better solutions are more likely to survive.

TSP (Travelling salesman problem) is a NP-hard problem, however, it’s possible for us to solve this problem with the help of GA. The following scatter plot is drawn according to the given dataset. So, the traveler is going to pick up the lowest cost route to travel all cities.

图表, 散点图

描述已自动生成

figure1: cities scatter plot

Figure2 shows the initial ordered route, which means the traveler travels all the cities in an ascending order (0->1->2->…->100->0).

图表

描述已自动生成

figure2: initial route

# Using GA to tackle simple TSP

First, I use permutation encoding method, thus, a chromosome is consisted of 101 genes (same as cities) and each gene is an integer from 0 to 100. The NumPy library gives us APIs to simply achieve this function, such as , .

Second, elitism and Roulette Wheel Selection are used in my implementation. For elitism, I keep 10% of the best chromosomes in the last generation and directly reintroduce them in next generation to replace 10% worst chromosomes in current generation.

In Roulette Wheel Selection, I calculate the percentage of each chromosome by its fitness.

Then generate a random number, judge which sector it falls in and take the corresponding chromosome as parent. By the way, the fitness is just the inverse of the cost of one route (chromosome).

Third, crossover is maybe the most important and difficult part in this algorithm. I don’t use crossover rate in this procedure. Because I think Elitism is introduced so it’s unnecessary to preserve a portion of parent chromosomes. Since each pair of parents can reproduce two children, I directly let children to replace parents.

I fulfill the algorithm we learnt on lecture and tutorial. Figure3 below shows the algorithm in a graphical way.

日程表

中度可信度描述已自动生成

figure3: crossover operator

And I’ll show the pseudo code of this algorithm:

(Hint: The start index is just the index of D in figure3, Slice is DE and Len is length of slice)

|  |
| --- |
| randomly generate Start Index, Len  intercept the retained part Slice  an Empty List Offspring  for gene in parent2:  if len (Offspring) == Start Index, append Slice into Offspring.  if gene is not in Slice, append it into Offspring.  else continue  let Offspring replace parent1 |

In practice, I generate two offspring simultaneously and replace the parents at the same time.

Finally, it’s mutation operator. It’s relatively easier than former two operators. Given a chromosome, generate a randomly number and compare it with mutation rate. If it’s smaller than the rate, then again randomly generate two indexes and exchange the genes according to the indexes.

The Following figures are experimenting results.

图表, 图示

描述已自动生成

图表, 折线图

描述已自动生成

figure4: fitness picture, epoch 5000, population 50 figure5: route picture, epoch 5000, population 50

we can see from the figure5, after genetic algorithm, the route picture is clearer and more comfortable than figure2. And the Fitness is not keep increasing since sometimes we should avoid staying in the local minimum. Figure6 and 7 uses 100 chromosomes to further optimize the problem, nevertheless, the time complexity also increases as population is growing bigger.

图表

描述已自动生成

图表, 折线图

描述已自动生成

Figure6: fitness picture, epoch 5000, population 100 figure7: route picture, epoch 5000, population 100

# Solution to Dynamic TSP

Since the GA itself is not so different with the problem 1. I’m not going to talk too much about it.

First, I add a “ResetDistanceMatrix” function, and execute this function to change the distance matrix every 100 epochs until e=5. Additionally, I rewrite the “RandomGenerateChromosome” function which inherited from the parent class in problem 1. Therefore, I can control the current population to satisfy the request proposed in problem while in the former task I just fix the number.

Second, I add a “ReuseChromosome” function to make use of the last event best chromosomes to make comparisons with the procedure that does not use last event best individuals. In this function, I reintroduce 5 best chromosomes in last event to the current event just like what I do in elitism.

Again, I put 4 pictures below to show you the difference between whether to use the last event best chromosomes or not. All the experiment parameters are the same, population is 50, epoch is 5000 and mutation rate is 0.3.

Figure8 and 9 is the result under the condition that we use the last event best solutions. Frankly speaking, I can’t see obvious acceleration between the above and below pictures. I guess maybe it’s because the problem set is not big enough or 100 epoch is too small that it couldn’t produce any excellent solutions in the last event.

图示

描述已自动生成

图表, 折线图

描述已自动生成

Figure8: fitness picture, epoch 5000, population 50(reuse) figure9: route picture, epoch 5000, population 50(reuse)

图示

描述已自动生成

图表, 折线图

描述已自动生成

Figure10: fitness picture, epoch 5000, population 50 figure11: route picture, epoch 5000, population 50

# Introduction to K-means

K-means is a widely used clustering technique. It’s a popular unsupervised learning algorithm and not very hard to fulfill.

The main idea is we randomly generate K centers at first. Then since the dataset tells us the X-Y coordination, we can calculate the Euclidean Distance between every city and every center. For a fixed city, we assign it to the nearest center. After finishing assigning all the cities, we calculate the average coordinate of the cities which are belong to the same center as the new center coordinate. So, keep looping until the center coordination does not change anymore or we can use iteration limits to control the loop which is more reliable and milder.

Figure12 and 13 shows the result after performing K-means with different K to 202 cities.

图表, 散点图

描述已自动生成

Figure12: K means with K = 5

图表, 散点图

描述已自动生成

Figure13: K means with K = 10

It’s not hard to realize what’s going on above, we just cluster those cities which are near to each other, and each cluster is represented by a unique color. If you’re carefully enough, you’ll find there’re markers like ‘X’ in the picture. Those are centers!

# Combine GA and K-means to address large-scale TSP

The basic procedure is that we perform GA on each cluster and finally combine all the clusters together. It seems extremely easy, but actually it’s a little bit formidable to make the combine operation in practice. **Here I refer a paper “Genetic Algorithms Based on Clustering for Traveling Salesman Problems”** which taches me how to combine such two Hamilton Circles.

The GA algorithm still doesn’t change except the “RandomGenerateChromosome” function. In this function, now I should generate chromosomes whose num of genes is determined by the num of cities in each cluster.

The vital important part is how to combine two Hamilton circles. As the paper described, we first choose two nearest points from two different clusters. Then, a great deal of choices needs to be considered. Let two points be , then we have which represent prior and posterior point of each point.

Assume one condition, breaks up the link with and breaks up the link with . Then we need to consider whether to link to or . To minimize the cost, the formula is just .

But be cautious that which link to break up is also unknow, so now you know you should consider the additional conditions like breaks up the link with and breaks up the link with . The following picture shows the algorithm in graphical way,

图片包含 不同, 各种, 充满, 许多

描述已自动生成

Figure14: combine Hamilton circles algorithm

When I code to simulate the process, it’s just more complicated than I imagine because we need to care about the list index problem (you can’t simply add 1 or minus one since it may exceed the length or be -1) and all these conditions, sometimes you need to think of the inverse situation and so on. So, in practice, I mainly write two functions named “ChooseDeleteAndInsert” and “MergeNodes” (in KMeansGA-LargeTSP python file) to fulfill the algorithm.

In this task, since we deal with one cluster each time, the total fitness is not available here, so I won’t show pictures about fitness. The following pictures are all about the routes with different K.

图示

描述已自动生成

Figure15: route picture, epoch 3000, mutation rate 0.2, K = 5

图示

描述已自动生成

Figure16: route picture, epoch 3000, mutation rate 0.2, K = 10

Actually, when I apply K-means to small problem like problem1, I can get better answers. So, I do think clustering technique is really a powerful weapon of GA.

# Brief introduction to multi-object optimization

In most of real-world situation, we can’t optimize all the objective functions at the same time. I think a classical problem is Bag-Problem in Computer Science which asks us to solve it with dynamic programming. Hence, our mission becomes finding a set of pareto solutions which consist of pareto front. Each solution on pareto front is not dominated by any other solution on PF.

# Using weighted-GA to solve problem4

So, in problem 4, the goal of the traveler is not only to minimize the route cost but also maximize the profit he can get.

Since two goals may be contradictory，it’s unlikely that we can find a unique optimizer achieving both two goals at the same time.

A simple approach is constructing a cost function by integrating both two goals. Here, the formula is , where is the route cost and is profit.

Obviously, if is very small (imagine that it’s going to be 0), then it’s same that we are just focusing on minimizing route cost. On the other hand, when is a very big number then what we’re doing is just to find a route that makes most profit.

In practice, I mainly make 3 changes. First is to add a function named “RandomGenerateProfit”, it’s easy to understand so I’m not going to talk too much about it.

Second is to modify my fitness function. Since it’s possible that our cost becomes a negative number, I can’t just take the inverse of the cost as the fitness. I use as my fitness function. For exponential function, I can ensure that fitness is a positive number. And is to give higher priority to negative cost than positive cost.

Third is to make a little bit change to the “TSPCost” function, we need to calculate the profit and minus it according to the formula.

Finally, I use while loop to get different solutions under different . Figure17 shows the pareto front.

图表, 散点图

描述已自动生成

Figure17: Pareto Front

# Introduction to NSGA-II

In the paper “A\_fast\_and\_elitist\_multiobjective\_genetic\_algorithm\_NSGA-II”, the author provides two brand new algorithms. One is “fast non dominated sort” and the other is “crowding distance” to preserve the diversity.

The first algorithm mainly helps us find those non-dominated solution quicklier. We need to use 2 arrays, one is to calculate how many times that the current solution is dominated, and another is to store the solutions that current solution dominates. After finishing the above procedure, we first pop up those solutions whose dominated num is 0 and decrease the dominated num by 1 for the solutions that are dominated by the popped up solution. Then keep the loop until we pop up all the solution and we can have a clear view that we need to give higher priority to those lower rank solutions.

The second algorithm is mainly to help how to choose individuals. According to the paper, we need to combine the old and the new generation together and choose best chromosomes. Because sometimes the solutions are too close to each other, so it’s not a good idea to take those individuals and plot them. Therefore, the author created an index to measure the closeness and our mission is to take those solutions whose closeness is relatively large.

The pseudo code is given in the paper, thus, I won’t repeat it. The following figure shows the Pareto front of problem 4 using NSGA-II. Since I only take 300 epochs, the range of route cost and profit is smaller than we get in figure17.

图表, 散点图

描述已自动生成

Figure18: Pareto Front

Now let me compare NSGA-II with weighted-GA and talk about the merits and disadvantages of each algorithm.

For weighted-GA, the main disadvantage is that it’s time-consuming. Since each pareto optimal solution takes one loop, it’s definitely a waste of time. Nonetheless, the main advantage is that I can get a more precise solution under a fixed .

The main advantage for NSGA-II is that we can get the pareto optimal solutions at the same time. And I think the main disadvantage is the time complexity (especially quick non dominated sort) and space complexity is higher than weight-GA. Additionally, I think the optimal solution calculated by NSGA-II is less precise than we get in weighted-GA.

# Using NSGA-II to solve multi-objective optimization

To tackle problem5, we just need to do some modifications to the parent class we write in problem4. The most important thing is to add a function to calculate the violation time so as to minimize it. Then it’s necessary to change the condition of dominance since we need to consider an extra factor. Finally, we need to draw 3 dimensions picture because there’re 3 objective functions.

Figure19 shows the result of problem5.

图表, 散点图

描述已自动生成

Figure19: Pareto Front

# Illustration to python code file structure and function of each file

Here, figure20 shows the file structure in my local IDE. And I’ll only submit the files which I’ve marked.

文本

描述已自动生成

Figure20: File Structure

First, I’ll introduce “utils” folder. It mainly contains 8 python files, they’re the core implementations of the algorithm I’ve talked in this report. There’re two files that may be confusing, which are “readfile.py” and “visualizeScatterPlot.py”. So, the first file is to change the structure of given dataset so that I can directly use library to extract data from it. Actually, the second file is not very difficult to understand. It contains some plot functions so that I can provide all the figure above.

Second, the “Code and Dataset” folder. When you execute the function provided in “readfile.py”, it’ll automatically generate a new structure dataset named “Nicer\_dataset\_gzx.txt” under the folder. And I want to claim that “gzx” is the abbreviation of my full name.

Eventually, the only left file is “stepptest.ipynb”, I just call all the functions here and get result here.