CSCI-561 – 2025 - Foundations of Artificial Intelligence Homework 1

Due February 10, 2025, 23:59:59 PT

1. Problem Description

This is a programming assignment in which you will apply AI search/optimization techniques to a 3D Travelling-Salesman Problem (TSP) such as the one shown in Figure 1. Conceptually speaking, the space of traveling is a set of "cities" located on some grid points with (x, y, z) locations in which your AI agent has to travel. Your agent can travel from any city to any city, and the distance between two cities is defined as the **Euclidean distance** between the two grid locations.

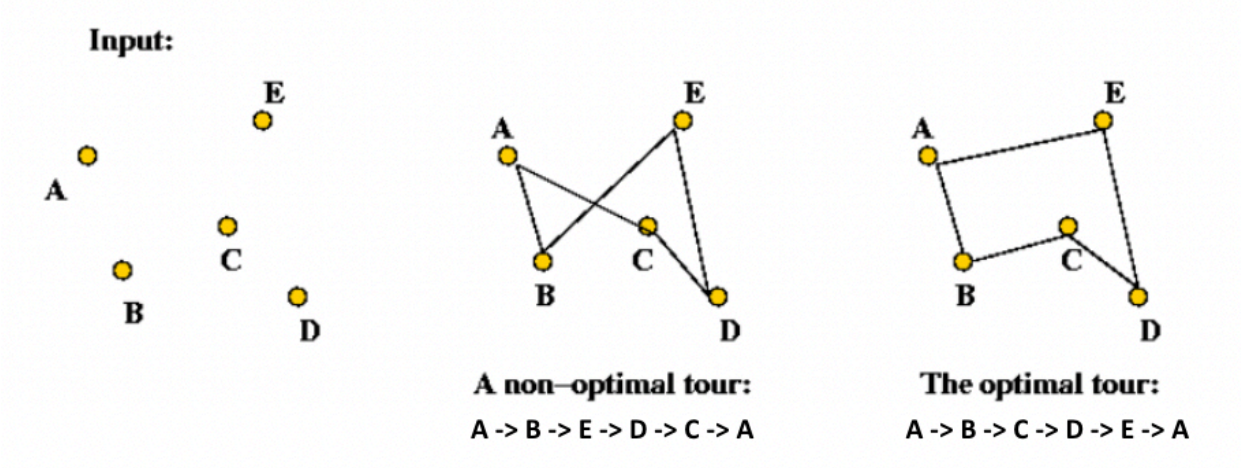
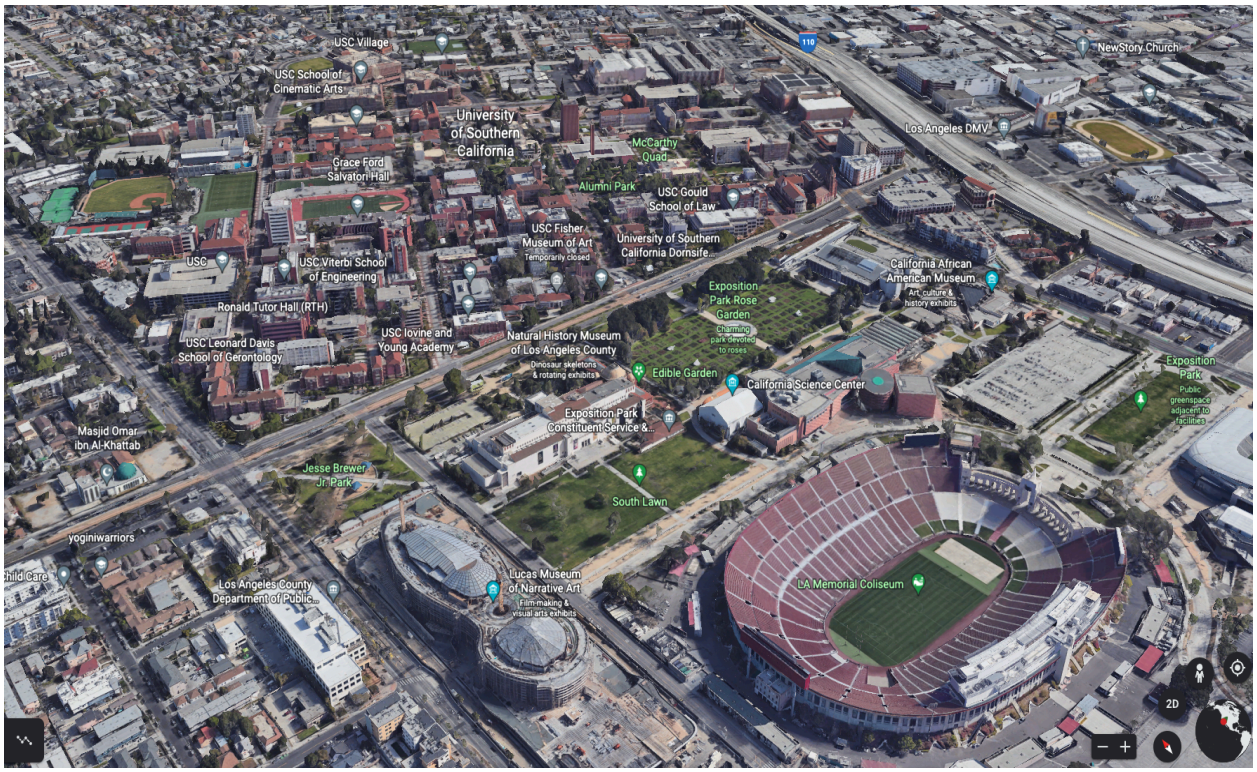


Figure 1: Traveling-Salesman Problem in 3D Space.

Your programming task is as follows: XYZ is a student at USC who has to run some errands across campus. The student starts from his house location, travels around USC to complete the errands, and returns home, or else the student will have to sleep on the streets :). A list of 3D coordinates where each of the coordinates points to a location within the USC campus is given (the z coordinate represents the floor of the building located at (x, y)). The problem is that the student is lazy and does not want to walk a lot. Being a Computer Science student, XYZ notices that this problem resembles the Traveling Salesman Problem (TSP) and wants to tackle this problem using one of the concepts he studied in CSCI-561, i.e., the **Genetic Algorithm**.

In TSP, given a list of cities/locations, the person has to **go to all the locations exactly once, return to the starting point**, and cover the minimum distance as a whole.

The student is happy that he had this class and knows that by using the **Genetic Algorithm**, he will be able to find a decently optimal path. But he forgot how the Genetic Algorithm works. Now, you, being a good friend of XYZ and a current student of CSCI-561 during Fall 2024, have to help your friend solve this problem. Your AI agent should try to find the path with the shortest distance, visit every location exactly once, and return to the start location. The shorter the path distance, the better your agent is.



2. Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe, you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

Do not copy code or written material from another student. Even single lines of code should not be copied.

Do not collaborate on this assignment. The assignment is to be solved individually.

Do not copy code off the web. This is easier to detect than you may think.

Do not share any custom test cases you may create to check your program's behavior in

more complex scenarios than the simplistic ones considered below.
Do not copy code from past students. We keep copies of past work to check for this. Even though this problem differs from those of previous years, do not try to copy the homework solutions from the previous years.
Do not post on Piazza asking how to implement some function for this homework or how to calculate something needed for this homework.
Do not post code on Piazza asking whether or not it is correct. This is a violation of academic integrity because it biases other students who may read your post.
Do not post test cases on Piazza asking what the correct solution should be.
Do ask the professor or TAs if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

3. Assignment Terminologies

Location: A location is represented as a combination of 3D coordinate points, x, y, and z, as shown in the figure above. For example, (10, 0, 30) represents a location with x= 10, y = 0, z= 30.

Path: A list of locations that your agent will visit in order **only once** and **return to the starting location**.
For example, if we have 3 locations, A, B, and C (each represented with 3D coordinates), then a valid path for this would be [A -> B -> C -> A]. Note that the path is a loop with the starting location (A) as the final endpoint.

4. Grading

Grading for this assignment consists of 2 parts:

Part A - Optimality with respect to TA's agent (60%)
We will run your search agent on the set of hidden grading cases to get the corresponding shortest paths. We will then use your path to calculate the path cost to grade your agent using the following criteria:

- Your score for a grading test case = (TA's agent shortest path cost)/(your-path cost)
- If your path cost < TA's agent path cost, we will give you full marks, i.e., 1 point for that grading test case.
- We then sum up all scores obtained and weigh it by 60%.

Part B - Quality of your solution (40%)
This section will test the quality of your agent against the agents developed by other students. Students need to research and come up with ways to improve their Genetic Algorithm.

The path scores obtained in Part A (above) will be ranked against other students' agents. A rank-based score will be calculated for the student's agent and contribute to 40% of the grade

for this assignment.
Student's score $= 40 * (1 - (rank/number\ of\ students))$

5. Tips for Implementation

The students can read through the following material in this section and use it as suggestions for their implementation.

NOTE: This section doesn't list the entire workings of the Genetic Algorithm and is in no way mandated to be followed exactly. These can be used as starting points for your assignment. **You are free to choose your implementation methods based on your research.**

a. Initial Population:

i. You will need to create the initial population, and you can do it as the following:
CreateInitialPopulation(size, cities): Arguments defined below:
Size: An integer representing the size of the list (initial_population) that needs to be returned.
Cities: A list of cities/locations where a location is represented in 3D coordinates (x,y,z)
Return Value: returns a list of paths (a permutation of cities), i.e., a list of lists, of size = size

```
def CreateInitialPopulation(size, cities):
    initial_population = []
    # your code goes here


    return initial_population
```

Students need to implement this function, which creates paths randomly or with some heuristic chosen by you.

b. Parent Selection:

i. In a genetic algorithm, parent selection is an important step. The method your agent implements to select a parent will determine the optimality of your solution.
CreateMatingPool(population, RankList): Arguments defined below:
Population: A list of paths from which the mating pool is to be created
RankList: A list of tuples of index and fitness scores sorted in descending order.
Return Value: A list of populations selected for mating (List contains paths)

Function Definition: This function defines the best-fit individuals and selects them for breeding. You will implement a **roulette wheel-based selection** (reference link), which is a widely used and efficient method for selecting parents. Make sure to assign the appropriate probabilities to the parents for roulette wheel selection.

```
def CreateMatingPool(population, RankList):
    matingPool = []
    # your code goes here


    return matingPool
```

c. Crossover:

i. Crossover(Parent1, Parent2, Start_index, End_index): Arguments are as follows:

Parent1: First argument of the function: A list containing the random sequence of cities for the salesman to follow

Parent2: Second argument of the function: A list containing the random sequence of cities for the salesman to follow

Start_index: Start index of the **SUBARRAY** to be chosen from parent 1

End_index: End index of the **SUBARRAY** to be chosen from parent 1

Return Value: Return child after performing the crossover (also a list containing a valid sequence of cities)

Function Definition: In this function, students are asked to implement a two-point crossover. Choose the subarray from Parent1 starting at start_index and ending at end_index. Choose the rest of the sequence from Parent 2. For example, if Start_index = 1 and End_Index = 3, then the child created should look like the following:

Parent-1: 1,2,3,4,5
Parent-2: 5,4,3,2,1
Start_index = 1
End_Index = 3
Resulting Parent 1 subarray from Start_index to End_index = 2,3,4

Your function should return the child as:
=> Child: 5,2,3,4,1

Note: For TSP, the child follows the constraint that each city is only visited once. So, any conflicts after copying the substring from parent 1 and the rest of the sequence from parent 2 should be resolved by your function. For example:

Parent-1: 1,2,3,4,5
Parent-2: 5,2,3,1,4

Start_index = 1
End_Index = 3
Resulting Parent 1 substring from Start_index to End_index = 2,3,4

Your function should return the child as:
=> Child: 5,2,3,4,1

Steps to resolve conflict:

- Copied subarray from parent-1 = 2,3,4
- Rest of the subarray from parent-2 = 5, . . .,4
- Resulting child list = 5,2,3,4,4
- Since the child doesn't follow the TSP constraint (city 4 appears twice), we replace the last 4 from parent 2 with the missing city, i.e., city 1.

```
def Crossover(Parent1, Parent2, Start_Index, End_Index):
    child = []
    # your code goes here

    return child
```

6. Input and Output

Your program will be run in a directory on Vocareum that contains the following input file. Your program should output the output file in the same directory.

Input: The file input.txt in the current directory of your program will be formatted as follows:

- 1st line: A strictly positive 32-bit integer N, indicating the number of "city" locations in the 3D space.
- Next N lines: Each line is a city coordinate represented by three non-negative 32-bit integers separated by one space character, for the grid location of the city.

Output: Report your path of traveling to the cities, that is, the total distance traveled and N locations of the cities.
For example:

- 1st line: Computed distance of the path.
- Next N+1 lines: Each line has three non-negative 32-bit integers separated by one space character indicating the city visited in order.
- **Note**: Your path ends at the start city. Hence, you will have N+1 lines.

For example, if there are five cities, then the output file would look something like:

```
8                    // computed distance of the path
```

60 103 97 // the location of the 1st city visited
61 103 97
62 103 97
63 103 97
64 103 97 // the location of the Nth city visited
60 103 97 // the location of the start city

To assist your programming, you will be provided with some sample inputs and outputs (see below). Please understand that the goal of these samples is to check that you can correctly parse the problem definitions and generate a correctly formatted output. The samples are very simple, and it should not be assumed that if your program works on the samples, it will definitely work on all hidden grading cases used for grading. There will be more complex cases, and it is your task to make sure that your program works correctly on any valid input. You are encouraged to design and try your own test cases to check how your program would behave in some complex special cases that you might think of. Since **each homework is checked via an automated AI script**, your output should match the specified format ***exactly***. Failure to do so will most certainly cost some points. The output format is simple, and examples are provided. You should upload and test your code on vocareum.com in their terminal window, which is a Linux-like environment. Please make sure you test your program in the terminal window at vocareum.com before you click the submit button there. You can submit as many times as you like, and the last submission before the deadline will be used to grade your work.

7. Notes and Hints

- You may use any of the following programming languages: **C++, Java, Python3**. However, Python is the preferred language for today's large-scale AI program applications.
- Please name your program "homework.xxx" where 'xxx' is the extension for the programming language you choose ("py" for Python, "cpp" for C++, and "java" for Java). If you are using C++11, then the name of your file should be "homework11.cpp", and if you are using Python3, then the name of your file should be "homework3.py". Please use the programming languages mentioned above for this homework.
- If you are using Python, then make a note that the version of Python that is offered is Python 3.7.5. Hence, the walrus operator and other features of the higher versions of Python are not supported.
- Try first to fully understand the Genetic Algorithm before developing your own code.
- To allow us to grade the whole class in a reasonable amount of time, your program will be killed after some time (e.g., it takes too long to return or appears stuck on a given test case. In this situation, you will get 0 points on the ongoing test case regardless of the fact that anything is generated or not. We will make sure that the time limit for a given grading case (or class) is sufficient and long enough to solve the case with a correct algorithm implementation.
- The time limit is the total combined CPU time as measured by the Unix time command. This command measures pure computation time used by your program and discards

time taken by the operating system, disk I/O, program loading, etc. Beware that it accumulates time spent in any of the threads spawned by your agent (so if you run 4 threads and use 400% CPU for 10 seconds, this will count as using 40 seconds of the allocated time).

- There is no limit on the input size, the number of cities, and any other 32-bit integer specified above. However, we will seriously consider the complexity of each case. You should take care of the data structures used in your algorithms so that the program returns in a bounded time. Additional information may be posted on Piazza if necessary. Please keep your eyes on it.
- If several equally-optimal solutions exist, then any of them will count as correct.
- There may be a lot of Q&A on Piazza. Please always search for relevant questions before posting a new one. Duplicate questions make everyone's lives harder.
- Only submit the source code files (in .java, .py, or .cpp) and helper files (if any, in .json or .txt). All other files should be excluded.
- Please submit your homework code through Vocareum (https://labs.vocareum.com/) under the assignment HW1. Your username is your email address. Click "forgot password" when logging in for the first time. You should have been enrolled in this course on Vocareum. If not, please post a private question with your email address and USC ID on Piazza so that we can invite you again.
- You can submit your homework code (by clicking the "submit" button on Vocareum) as many times as you want. Only the latest submission will be considered for grading. After the initial deadline, the submission window will still be open for five days. However, a late penalty will be applied as 20% per day if your latest submission is later than the initial deadline.
- Every time you click the "submit" button, you can view your submission report to see if your code works. The grading report will be released after the homework's due date.
- You don't have to keep the page open on Vocareum while the scripts are running.
- Every time you click the "submit" button on Vocareum, your submitted agent will be run and tested by our AI script on a number of test cases, and the results will be reported to you in the submission report, which you can read and examine. You may use these reports to debug and improve your agent. Notice that these are "SUBMISSION cases" and not "GRADING cases". The grading cases are reserved for grading purposes and may not be available to your search agent before the grading process begins.
- It's highly suggested that you reserve some time for submission and testing on Vocareum because you may come across some technical issues if this is your first time using it. Anyway, don't wait until the last minute!
- Be careful and avoid multiple submissions of large files to Vocareum. Vocareum does not allow students to delete old submissions, and in the past, students ran out of space and were unable to use Vocareum until we got in touch with support and asked them to delete files.

8. Grading Methods

Your code will be tested and graded as follows: Your program should not require any command-line argument. It should read a text file called “input.txt” in the current directory that contains a problem definition. It should create and write a file “output.txt” with your solution in the same current directory. The format for input.txt and output.txt is specified in section 5. **The end-of-line character is LF** (since Vocareum is a Unix system and follows the Unix convention).

The grading AI script will test your program for 40 test cases for grading as follows:

- Create an input.txt file and delete any old output.txt file.
- Run your code to create your output.txt file.
- Check the correctness of your program’s output.txt file.
- For Section 4, Part A, there will be 40 test cases divided into four classes: Class 1 (easy), 10 cases; Class 2 (medium), 10 cases; Class 3 (hard), 10 cases; and Class 4 (complex), 10 cases.
- We will make sure that the time limit for a given grading case (or class) is sufficient and long enough to solve the case with a correct algorithm implementation.
    - Class 1 (easy): < 60 seconds
    - Class 2 (medium): < 75 seconds
    - Class 3 (hard): < 120 seconds
    - Class 4 (complex): < 300 seconds

  Time limits are uniformly measured by Vocareum (instead of anyone’s local machine). They are typically determined by a standard algorithm implementation that has been tested thoroughly, and we ensure that these time limits are typically very generous.

Note that if your code can’t be compiled, or somehow fails to load and parse input.txt, or writes an incorrectly formatted output.txt, or no output.txt at all, or OuTpUt.TxT, **you will get zero points**. Anything you write to stdout or stderr will be ignored and is ok to be left in the code you submit (but it will likely slow you down). Please test your program on Vocareum’s terminal window with the provided sample files to avoid any problems.

9. Example Input and Output

The students need to understand the format in which the input.txt is given and the format of the output.txt that is needed from them. Post your queries on Piazza if you have any.

Example 1:

=========input.txt==========

```
3
158 147 135
56 24 160
162 194 104
```

==========

=========output.txt=========
426.198
162 194 104
158 147 135
56 24 160
162 194 104
==========

Example 2:

=========input.txt==========
5
59 170 117
113 152 44
174 135 162
11 36 174
177 32 24
==========
=========output.txt=========
746.671
11 36 174
174 135 162
177 32 24
113 152 44
59 170 117
11 36 174
==========

Example 3:

=========input.txt==========
7
199 173 30
120 199 34
144 39 130
137 199 93
153 196 97
175 53 76
173 101 186
==========
=========output.txt=========
576.125
120 199 34
199 173 30
175 53 76
144 39 130
173 101 186
153 196 97
137 199 93
120 199 34
==========