CS512 FUN Projects - Path Recommending System

Xu Liu
Rutgers University
Piscataway, NJ, USA
Email: xu.liu.bupt@rutgers.edu

Xinghao Wang
Rutgers University
Piscataway, NJ, USA
Email: xinghao.wang@rutgers.edu

Tianfang Zhang
Rutgers University
Piscataway, NJ, USA
Email: tz203@scarletmail.rutgers.edu

***Abstract*— This article mainly introduces the achievement of a path recommending system. This system is a handy tool for every user who wants to find a path with minimal length. The problem is actually a TSP(Traveling Salesman Problem). In this system, we use approximation method-Genetic Algorithm to solve it.**

I. Project Description

The type of this project is implementation of an algorithm not covered in class. There are a lot of specific approximation algorithms . This system uses genetic algorithms to find an optimal path. Path planning plays an important part in our daily life, especially for people who are planning trips and want to find a path with minimal length, therefore developing a path recommending system helps a lot and is extremely useful. Using genetic algorithms, we can provide a path for users as well as possible. Each one in our group is proficient in data structures and algorithms, so it is feasible for us to complete it within a semester. By using approximation method, we may not be able to get an optimal path, but its efficiency is excellent. This is the novel part of this system. Until now, we have not encountered any stumbling block.

The project has four stages: Gathering, Design, Infrastructure Implementation, and User Interface.

A. Stage1 - The Requirement Gathering Stage.

- The general system description : Assuming that there is a person who wants to take a trip starting from his hometown to several places by car, and he wants to find a path with minimal length. This project is a path recommending system which is able to provide users with a path as short as possible.
- The three types of users (grouped by their data access/update rights):
  1) **Administrator**: Administrators are able to manage database, manage user roles.
  2) **Normal users**: Mostly the people who are planning a trip, normal users can use the system to get a path as short as possible.
  3) **System maintenance personnel**: System maintenance personnel is responsible for the maintenance of the system.
- The user's interaction modes: A user typically use a keyboard to input information, and uses mouse clicks to interact with the UI path recommending system. Generally, users only need to input their start point and several destinations in the specific areas, and click the search button, then the system will show the recommended path to them.
- The real world scenarios:
  - Scenario1 description: General users, mostly people who are planning a trip, and they wants to know the shortest path passing all their destinations.
  - System Data Input for Scenario1: Start point and destinations
  - Input Data Types for Scenario1: integer
  - System Data Output for Scenario1: Recommended path
  - Output Data Types for Scenario1: integer
  - Scenario2 description: General users want to know about the complete route map.
  - System Data Input for Scenario2: Route map
  - Input Data Types for Scenario2: String
  - System Data Output for Scenario2: A graph showing the complete route map
  - Output Data Types for Scenario2: A graph
  - Scenario3 description: A system maintenance personnel who wants to add new information.
  - System Data Input for Scenario3: Start point, destination and its distance
  - Input Data Types for Scenario3: String, integer
  - System Data Output for Scenario3: A message indicating whether the operation is successful or not
  - Output Data Types for Scenario3: String
  - Scenario4 description: A system maintenance personnel who wants to modify information.
  - System Data Input for Scenario4: Start point, destination and its distance
  - Input Data Types for Scenario4: String, integer
  - System Data Output for Scenario4: A message indicating whether the operation is successful or not
  - Output Data Types for Scenario4: String
  - Scenario5 description: A administrator who wants to add a new system maintenance personnel.
  - System Data Input for Scenario5: Username and password
  - Input Data Types for Scenario5: String
  - System Data Output for Scenario5: A message indicating whether the operation is successful or not
  - Output Data Types for Scenario5: String

- Scenario6 description: A administrator who wants to delete a system maintenance personnel.
- System Data Input for Scenario6: Username and password
- Input Data Types for Scenario6: String
- System Data Output for Scenario6: A message indicating whether the operation is successful or not
- Output Data Types for Scenario6: String

- Project Time line and Divison of Labor.
  - Stage 1: Before Nov. 5
    - Tasks of Xu Liu:
      - Format designing using LaTeX
      - Writing general description of this project
      - Writing the timeline and division of the project
    - Tasks of Xinghao Wang:
      - Writing five real-world scenarios of this system
    - Tasks of Tianfang Zhang:
      - Writing 3 types of users of this system
      - Writing the user's interaction modes of this system
  - Stage 2: Before Nov. 24
    - Tasks of Xu Liu:
      - Writing a brief description of algorithms and data structures
      - Drawing flow diagram
    - Tasks of Xinghao Wang:
      - Writing a short textual project description
      - Writing flow diagram major constraints
    - Tasks of Tianfang Zhang:
      - Writing high-level pseudo code
  - Stage 3: Before Dec. 8
    - Tasks of Xu Liu:
      - Implementing the system
    - Tasks of Xinghao Wang:
      - Testing and evaluating the system
    - Tasks of Tianfang Zhang:
      - Writing documentation
  - Stage 4: Before Dec. 13
    - Tasks of Xu Liu:
      - Writing a project report
    - Tasks of Xinghao Wang:
      - Designing a GUI
    - Tasks of Tianfang Zhang:
      - Preparing for a power point presentation

B. Stage2 - The Design Stage.

- Short Textual Project Description.
  By analysis, we will actually design a system to solve the traveling salesman problem. Considering its time complexity, it is hard to provide a shortest path for the user. Therefore, we choose to use approximation algorithms to find a path as short as possible. What we finally choose is Genetic Algorithm.
  There are some definitions in Genetic Algorithm[1]:
  Individual: a single route satisfying the conditions of the Traveling Salesman Problem.
  Population: a collection of possible routes (collection of individuals).
  Parents: two routes that are combined to create a new route.
  Mating pool: a collection of parents that are used to create our next population (thus creating the next generation of routes).
  Fitness: a function that tells us how good each route is.
  Mutation: a way to introduce variation in our population.
  The genetic algorithm will first create the population, which means create a huge number of individuals and then compute the fitness of the population. Then it will select the mating pool in the population to choose proper parents to breed new individuals. Meanwhile, it will try to introduce variation into individuals in a random way. After repeating this process for some time, there will be some answer which match the request of fitness threshold.
- Flow Diagram.

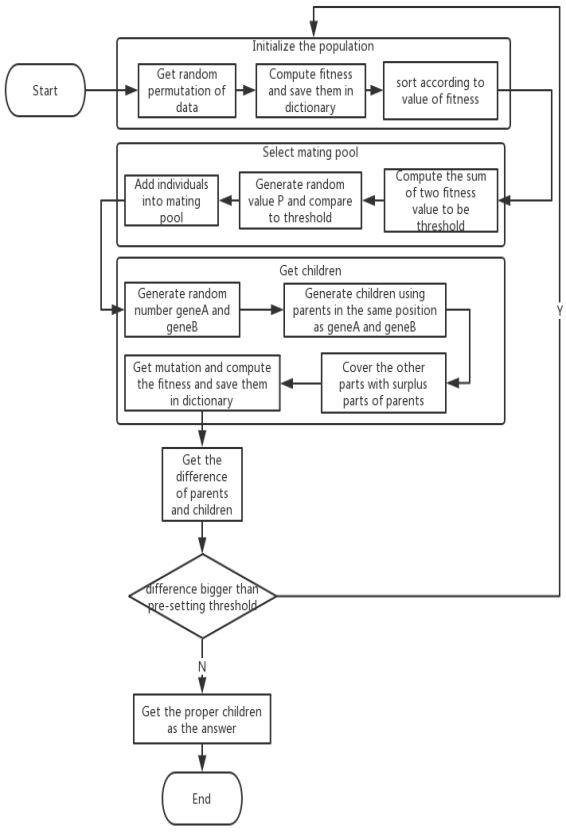


Fig. 1. Flow Diagram

- High Level Pseudo Code System Description[2].

Algorithm 1: Genetic Algorithm

Input : data: id and coordinates of all the locations except start location and end location,
size: size of population
Result: return the approximated optimal solution
population = initial_population(data, size);
children = get_children(population, size);
children = population ∪ children;
sort each child in children according to the fitness in decreasing order;
children = children[0:size];
dif = difference(population, children);
while *dif* ≥ *threshold* **do**
  population = children;
  children = get_children(population, size);
  children = population ∪ children;
  sort each child in children according to the fitness in decreasing order;
  children = children[0:size];
  dif = difference(population, children);
end
return Children[0];

Note:In this algorithm, the inputs are coordinates of locations. However, we will first build a map between the location's name and its coordinates. So, users only need input the location's name and the algorithm can translate it into the coordinates according to the map.

Algorithm 2: initial_population

Input : data: id and coordinates of all the locations except start location and end location,
size: size of population
Result: return the initialized population
population = empty list;
for *i=0* **to** *size* **do**
  single = random permutation of data;
  fit = fitness(single);
  pair = (single, fit);
  population = population ∪ pair;
end
sort population according to the fitness in decreasing order;
return population;

Algorithm 3: fitness

Input : target: a specific permutation of data
Result: return the fitness of the target
total_distance = 0;
for *i=0* **to** *len(target)-2* **do**
  distance = Euclidean distance of target[i] and target[i+1];
  total_distance += distance;
end
return 10000 / total_distance;

Algorithm 4: selection

Input : generation: a list of the breeding generation
Result: return the list of two selected parents
population = generation.copy();
parents = empty list;
for *k=1* **to** *2* **do**
  total_fitness = compute the sum of fitness in population;
  for *i=0* **to** *len(population)-1* **do**
    population[i].fitness = population[i].fitness / total_fitness;
  end
  thresholds = empty list;
  for *i=1* **to** *len(population)-1* **do**
    threshold = population[i].fitness + population[i-1].fitness;
    thresholds = thresholds ∪ threshold;
  end
  p = random value from 0 to 1;
  for *i=0* **to** *len(thresholds)-1* **do**
    if *p* ≤ *thresholds[i]* **then**
      parents = parents ∪ population[i];
      Delete population[i] in population;
      break;
    end
  end
end
return parents;

Algorithm 5: get_children

```
Input : population: a list of the breeding generation
        size: size of population
Result: return a list of generated children
children = empty list;
while len(children) < size do
    parents = selection(population);
    geneA = random integer between 0 and
    len(parents[0])-1;
    geneB = random integer between 0 and
    len(parents[0])-1;
    start = min(geneA, geneB);
    end = max(geneA, geneB);
    child1P1 = empty list;
    child1P2 = empty list;
    child2P1 = empty list;
    child2P2 = empty list;
    for i=start to end do
        child1P1 = child1P1 ∪ parents[0][i];
        child2P1 = child2P1 ∪ parents[1][i];
    end
    foreach item in parents[1] do
        if item not in child1P1 then
            child1P2 = child1P2 ∪ item;
        end
    end
    foreach item in parents[0] do
        if item not in child2P1 then
            child2P2 = child2P2 ∪ item;
        end
    end
    child1 = child1P1 ∪ child1P2;
    child2 = child2P1 ∪ child2P2;
    child1mutation = mutation(child1.copy(), 0.3);
    child2mutation = mutation(child2.copy(), 0.3);
    fit1 = fitness(child1mutation);
    fit2 = fitness(child2mutation);
    children = children ∪ (child1mutation, fit1) ∪
    (child2mutation, fit2);
end
return children;
```

Algorithm 6: difference

```
Input : population: a list of the parent generation
        children: a list of the child generation
Result: return the difference of two generations
fit1, fit2 = 0;
for i=0 to len(population)-1 do
    fit1 += population[i].fitness;
    fit2 += children[i].fitness;
end
return fit2 - fit1;
```

Algorithm 7: mutation

```
Input : target: the target child waiting to mutate
        rate: the mutation rate
Result: return the mutated target
p = random value between 0 and 1;
if p < rate then
    position1 = random integer between 0 and
    len(target)-1;
    position2 = random integer between 0 and
    len(target)-1;
    swap target[position1] with target[position2];
end
return target;
```

- Algorithms and Data Structures.
  How to build genetic algorithm of this problem[2]:
  1) Initial the population:We first get random permutation of the data in this problem, then compute its fitness and save them in the specific data structure. Finally, use sort algorithm to make it in order according to its fitness value.
  2) Compute the fitness value:To compute the value of fitness, we choose to compute the Euclidean distance between every two nodes in the permutation.
  3) Select mating pool:Use threshold to be the sum of first two fitness value of individuals and then generate random value p, compare it to the threshold and if it is smaller, add the first individual into mating pool.
  4) Get children:First, select the mating pool and generate random number geneA and geneB. Then from geneA to geneB, generate the children using the parts of parents in the same position. Then cover the other parts with the surplus parts of parents. Merge these two parts to generate two new children and using mutation function to make sure there are some mutation in children. Compute the new fitness and save them in the data structure of children.
  Some extra function:
  1) Difference function:Compute the fitness difference of parents and children.
  2) Mutation function:Generate a random value p and using probability p to judge whether there are parts swapping in two positions, which means there are some mutations in these positions.
  Data Structure Design:In this algorithm, we use a specific data structure to save individuals and its fitness values. Then use normal array or list to save other information and values.
- Flow Diagram Major Constraints.
  - Space Constraint
    Description: In our project, we will use genetic algorithm to make an approximation of the answer of traveling salesman problem. Because there is a large population of parents, it will cost so much spaces to save individuals and fitness values.
    Justification: When modifying our project and algorithm, we can use data structure will higher ef-

ficiency.

- Correctness Constraint
  Description: In our project, we only use approximation algorithm to get an approximate answer and, in many cases, the answer is not the exact answer.
  Justification: When modifying our project and algorithm, we can choose better approximate algorithm.

C. Stage3 - The Implementation Stage.

We are using Python 3.7 as our programming languange for this project. The programming environment is as follows:

a) Operating System: Windows 10 64-bit
b) Processor: Intel Core i5-9400F CPU
c) Memory: 16 GB DDR4-2666 Memory
d) Graphics Card: RTX 2060

The deliverables for this stage include the following items:

- Sample small data snippet.
  Start Points:1
  Stop Points:2,3,4,5,6,7,8,9,10
  Note:the map of number to the cities is in the following table:

Squence Number	City	Squence Number	City	Squence Number	City	Squence Number	City
1	AL	13	IA	25	NE	37	RI
2	AZ	14	KS	26	NV	38	SC
3	AR	15	KY	27	NH	39	SD
4	CA	16	LA	28	NJ	40	TN
5	CO	17	ME	29	NM	41	TX
6	CT	18	MD	30	NY	42	UT
7	DE	19	MA	31	NC	43	VT
8	FL	20	MI	32	ND	44	VA
9	GA	21	MN	33	OH	45	WA
10	ID	22	MS	34	OK	46	WV
11	IL	23	MO	35	OR	47	WI
12	IN	24	MT	36	PA	48	WY

Fig. 2. Map Table

- Sample small output
  Path:1,9,3,5,2,4,10,6,7,8,1
- Working code
  The code of GeneticAlgorithm:

```
from random import randrange, random, shuffle
from copy import deepcopy
import math


class GA:
    def __init__(self, size, p, tsp):
        self.tsp = tsp
        self.size = size
        self.mutation_rate = p
        self.iteration = 1000

    def genetic_alg(self):
        population = self.initial_population()
        children = self.get_children(population.
        copy())
        children = population + children
        children.sort(key=(lambda x: x[1]),
        reverse=True)
        children = children[0:self.size]
        dif = self.difference(population,
        children)

        while dif > 10:
            population = children
            children = self.get_children(
            population.copy())
            children = population + children
            children.sort(key=(lambda x: x[1]),
             reverse=True)
            children = children[0:self.size]
            dif = self.difference(population,
            children)
            print(dif)
        return children[0]


    def genetic_alg_iteration_based(self):
        population = self.initial_population()
        children = self.get_children(population.
        copy())
        children = population + children
        children.sort(key=(lambda x: x[1]),
         reverse=True)
        children = children[0:self.size]
        for i in range(self.iteration):
            population = children
            children = self.get_children(
            population.copy())
            children = population + children
            children.sort(key=(lambda x: x[1]),
             reverse=True)
            children = children[0:self.size]
            if i % 100 == 0:
                print("iteration", i, ' path:',
                 children[0][0])
                print("distance:", self.
                compute_length(children[0][0]))
        return children[0]


    def fitness(self, target):
        start = self.tsp.start
        total_dist = 0
        total_dist += math.sqrt(pow(self.tsp.
        data[start][0] -
        self.tsp.data[target[0]][0], 2) + \
        pow(self.tsp.data[start][1] -
        self.tsp.data[target[0]][1], 2))
        for i in range(len(target) - 1):
            former = target[i]
            latter = target[i + 1]
            dist = math.sqrt(pow(self.tsp.data
            [former][0] -
            self.tsp.data[latter][0], 2) + \
            pow(self.tsp.data[former][1] -
            self.tsp.data[latter][1], 2))
            total_dist += dist
        total_dist += math.sqrt(pow(self.tsp.
        data[target[-1]][0] - self.tsp.data
        [start][0], 2) + \
        pow(self.tsp.data[target[-1]][1] -
        self.tsp.data[start][1], 2))
        return 100000 / total_dist


    def compute_length(self, target):
        start = self.tsp.start
        total_dist = 0
        total_dist += math.sqrt(pow(self.tsp.
```

```
            data[start][0] - self.tsp.data[target
            [0]][0], 2) + \
             pow(self.tsp.data[start][1] -
             self.tsp.data[target[0]][1], 2))
            for i in range(len(target) - 1):
                former = target[i]
                latter = target[i + 1]
                dist = math.sqrt(pow(self.tsp.data
                [former][0] - self.tsp.data
                [latter][0], 2) + \
                pow(self.tsp.data[former][1] -
                self.tsp.data[latter][1], 2))
                total_dist += dist
            total_dist += math.sqrt(pow(self.tsp.


            [target[-1]][0] - self.tsp.data
            [start][0], 2) + \
            pow(self.tsp.data[target[-1]][1] -
            self.tsp.data[start][1], 2))
            return total_dist


        def initial_population(self):
            population = []
            for i in range(self.size):
                single = self.tsp.stops.copy()
                shuffle(single)
                fit = self.fitness(single)
                population.append((single, fit))
            population.sort(key=(lambda x: x[1]),
             reverse=True)
            return population


        def selection(self, population):
            chosen = []
            for k in range(2):
                total_fit = 0
                for i in range(len(population)):
                    total_fit += population[i][1]
                category = [population[0][1] /
                total_fit]
                for i in range(1, len(population)):
                    category.append(category[i - 1]
                     + population[i][1] / total_fit)
                rand = random()
                for i in range(len(category)):
                    if rand < category[i]:
                        chosen.append(population[i])
                        population.pop(i)
                        break
            return chosen


        def get_children(self, population):
            children = []
            while len(children) < self.size:
                parent1, parent2 = self.selection(
                deepcopy(population))
                pos1 = randrange(0, len(parent1))
                pos2 = randrange(0, len(parent1))
                start = min(pos1, pos2)
                end = max(pos1, pos2)
                child1_p1, child1_p2, child2_p1,
                 child2_p2 = [], [], [], []
                for i in range(start, end):
                    child1_p1.append(parent1[0][i])
                    child2_p1.append(parent2[0][i])
                for gene in parent2[0]:
                    if gene not in child1_p1:
                        child1_p2.append(gene)
                for gene in parent1[0]:
                    if gene not in child2_p1:
                        child2_p2.append(gene)
                child1 = child1_p1 + child1_p2
                child2 = child2_p1 + child2_p2
                child1_mut = self.mutation(deepcopy
                (child1))
                child2_mut = self.mutation(deepcopy
                (child2))
                fit1 = self.fitness(child1_mut)
                fit2 = self.fitness(child2_mut)
                children.append((child1_mut, fit1))
                children.append((child2_mut, fit2))
            children.sort(key=(lambda x: x[1]),
             reverse=True)
            return children


        def difference(self, old_set, new_set):
            n1, n2 = 0, 0
            for i in range(len(old_set)):
                n1 += old_set[i][1]
                n2 += new_set[i][1]
            return n2 - n1

        def mutation(self, target):
            rand = random()
            if rand < self.mutation_rate:
                pos1 = randrange(0, len(target) - 1)
                pos2 = randrange(0, len(target) - 1)
                temp = target[pos1]
                target[pos1] = target[pos2]
                target[pos2] = temp
            return target
```

The code of the test:

```
from GeneticAlgorithm import GA

class TSP:
    def __init__(self, data):
        self.data = data
        self.start = None
        self.stops = []

    def set_start(self, start):
        self.start = start

    def set_stops(self, stops):
        self.stops = stops

def data_load():
    data = {}
    with open('us48capitals.txt', 'r') as f:
        for line in f:
            lst = list(map(int,
            line.split(' ')))
            data[str(lst[0])] = (lst[1], lst[2])
    return data

data = data_load()
input_stops = [str(i) for i in range(2, 49)]
start = '1'
tsp = TSP(data)
tsp.set_start(start)
tsp.set_stops(input_stops)
approx_ga = GA(100, 0.1, tsp)
# best = approx_ga.genetic_alg()
best = approx_ga.genetic_alg_iteration_based()
path = best[0]
path.insert(0, start)
path.insert(len(input_stops)+1, start)
print('path:', path, 'total distance:',
 approx_ga.compute_length(best[0]))
```

- Demo and sample findings
  - Data size: We use capitals of 48 states in US as our data. And we first map the location of these capitals into corresponding coordinates. The size of data is only 1KB. The data can be found in us48capitals.txt.
  - The accuracy of this program heavily depends on the input. If we input several stop points, it is easy to find an excellent path.However, if there are many stop points, the result is a bit worse but acceptable.

D. Stage4 - User Interface.

Please insert your deliverables for Stage4 as follows:

- The initial statement to activate your application with the corresponding initial UI screenshot
  By running the program login.py, we can jump to the login page shown as Fig.3. All the user IDs and passwords are stored in the info.txt in the root directory. (User ID:123 Password:123 and User ID: admin Password: admin. We can add new user by modify the content of info.txt). After we enter correct User ID and Password, the main GUI will pop out shown as Fig.4 .

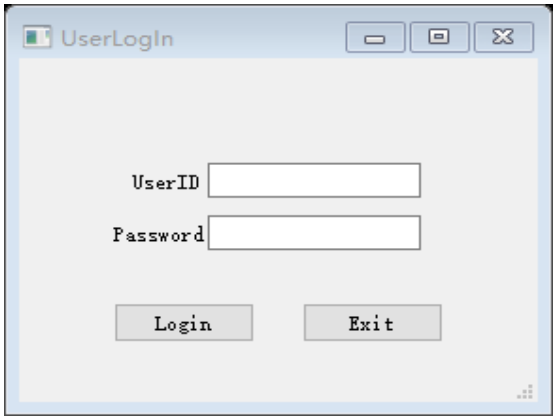


Fig. 3. Login Page

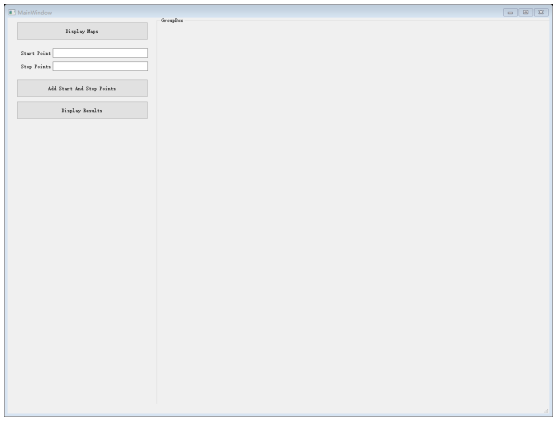
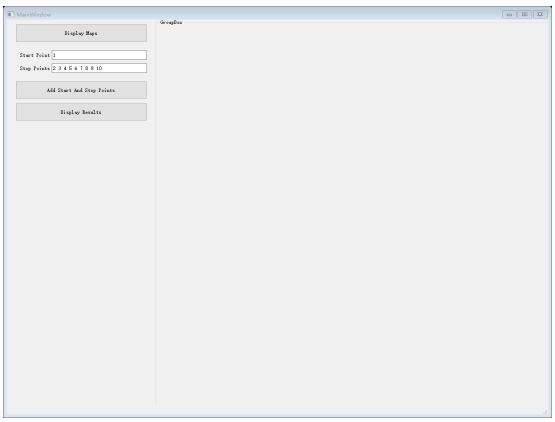


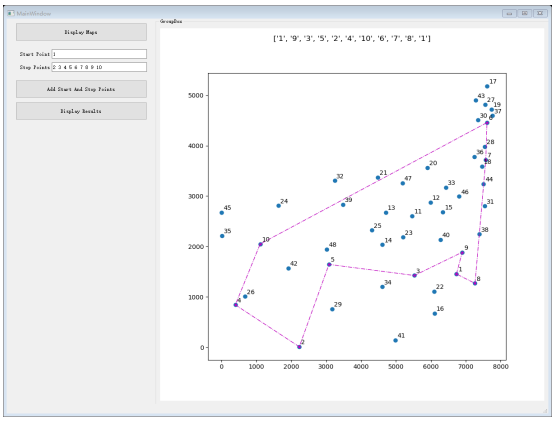
Fig. 4. Main GUI

- Two different sample navigation user paths through the data exemplifying the different modes of interaction and the corresponding screenshots.

1) A user inputs text in the input area using keyboard.



2) A user clicks buttons or clicks items in the list using a mouse.



- The error messages popping-up when users access and/or updates are denied (along with explanations and examples):
  - The error message: UserID or password error!
  - The error message explanation (upon which violation it takes place): If the input of user ID and password cannot match, we will get the hint 'User ID or password error!'
  - The error message example according to user(s) scenario(s):

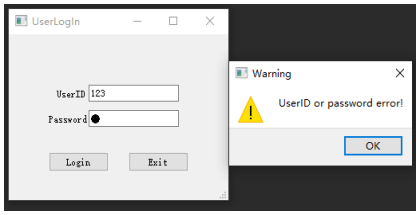


Fig. 5. UserID or password error!

- The information messages or results that pop-up in response to user interface events.
  - The information message: 'Wrong input of points'
  - The information message explanation and the corresponding event trigger : If we input the wrong format of points, such as 'A' and '50', which does not exist in the set of points, we will get the hint 'Wrong input of points'.The correct inputs are numbers from 1 to 48 as shown in Fig 2.
  - The error message example in response to data range constraints and the coresponding user's scenario

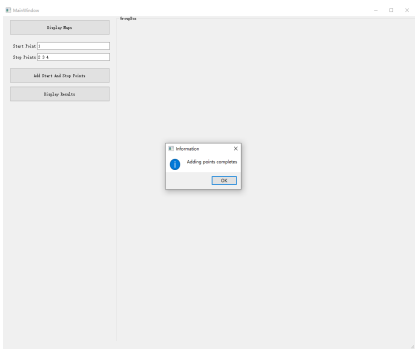


Fig. 6. 'Wrong input of points'

- The interface mechanisms that activate different views.
  - The interface mechanism: After the main interface launches, users can input numbers in the input area and click buttons. If a user clicks the 'add Start and Stop Point' button, the program will checks if the user's input is valid. If it is valid, it will show "Adding points completes". Then the user can clicks the "Display Results" button. After that, the program will show the path and the path graph in the right side.

II. Project Highlights.

- This project is actually a TSP. Considering time complexity, we use approximation algorithm(Genetic Algorithm) to solve this problem. We can input multiple stop points every time and finally get a good path solution using the approximation method.

References

[1] Koza J R. Genetic programming[J]. 1997.
[2] Potvin, Jean-Yves. "Genetic algorithms for the traveling salesman problem."Annals of Operations Research 63.3 (1996): 337-370