16350 Final Project Report

Problem Description

Optimization in Travelling Salesman Problem for Single Omni-directional Robot

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city? Send a robot on its way. He can move in all directions.

How To Run The Code

1. **Compile and Run**

   In terminal:

   ```
   g++ -std=c++14 bco.hpp bco.cpp -o bco.out
   ./bco.out
   ```

   In the case of 10 cities, the program takes about 2s to run. When it's done, the sequence of cities to visit will automatically be written into the file `cities.m`, which will be imported into MatLab later. In the terminal, three things would be printed: a matrix representing distances between any city i to j, the sequence of cities and the length of the generated path.

2. **Visualization**

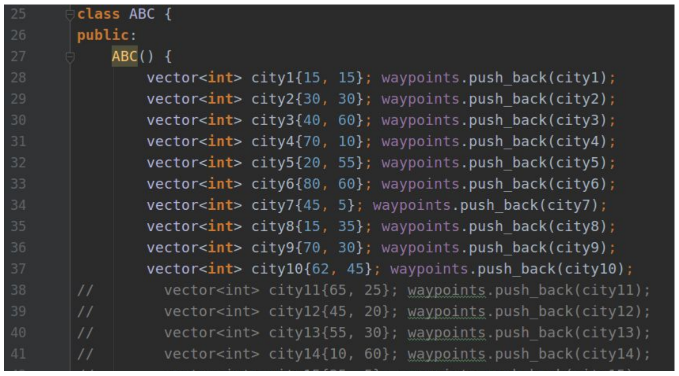
   Open matlab, navigate to the code folder and enter the following commands:

   ```
   mex planner.cpp
   cities
   run(citiesList)
   ```

   The cities will be labelled on the map 1 through 10 in the order which they will be visited by the robot. The robot will go through them one by one according to the sequence generated in step 1.

3. **Customize your map**

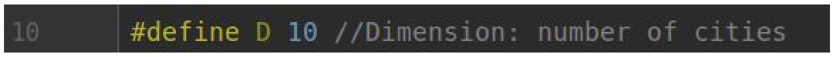
   Currently the program is optimized for 10 cities and the locations of cities are hardcoded. But you can also build the map in whatever way you like! At line 28 in bco.hpp, a list of cities are defined here. You may change their locations, uncomment city11-20, or add your own cities in a similar fashion, then repeat step 1 and 2 to compile and run.

   ```
   25  class ABC {
   26  public:
   27      ABC() {
   28          vector<int> city1{15, 15}; waypoints.push_back(city1);
   29          vector<int> city2{30, 30}; waypoints.push_back(city2);
   30          vector<int> city3{40, 60}; waypoints.push_back(city3);
   31          vector<int> city4{70, 10}; waypoints.push_back(city4);
   32          vector<int> city5{20, 55}; waypoints.push_back(city5);
   33          vector<int> city6{80, 60}; waypoints.push_back(city6);
   34          vector<int> city7{45, 5}; waypoints.push_back(city7);
   35          vector<int> city8{15, 35}; waypoints.push_back(city8);
   36          vector<int> city9{70, 30}; waypoints.push_back(city9);
   37          vector<int> city10{62, 45}; waypoints.push_back(city10);
   38  //        vector<int> city11{65, 25}; waypoints.push_back(city11);
   39  //        vector<int> city12{45, 20}; waypoints.push_back(city12);
   40  //        vector<int> city13{55, 30}; waypoints.push_back(city13);
   41  //        vector<int> city14{10, 60}; waypoints.push_back(city14);
   ```

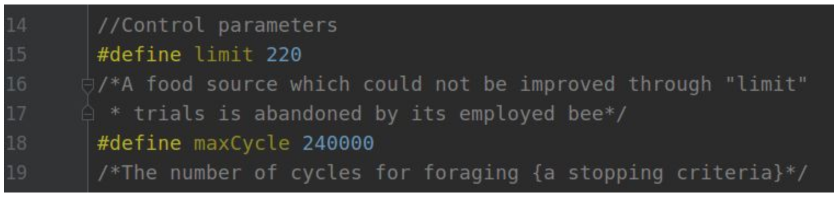
   IMPORTANT NOTE: (on the next page)

a. If you change the number of cities (currently set to 10), change the macro D on line 10 in bco.hpp.



```
10      #define D 10 //Dimension: number of cities
```

b. The current control parameters works well for equal or less than 10 cities. If you wish to have more, increase limit and maxCycle on line 15 and 18 of bco.hpp according to the first chart in section “Performance Evaluation”.



```
14      //Control parameters
15      #define limit 220
16      /*A food source which could not be improved through "limit"
17       * trials is abandoned by its employed bee*/
18      #define maxCycle 240000
19      /*The number of cycles for foraging {a stopping criteria}*/
```

Algorithm: Artificial Bee Colony Optimization

In the ABC model, the colony consists of three groups of bees: employed bees, onlookers and scouts. It is assumed that there is only one artificial employed bee for each food source. In other words, the number of employed bees in the colony is equal to the number of food sources around the hive. Employed bees go to their food source and come back to hive and dance on this area. The employed bee whose food source has been abandoned becomes a scout and starts to search for finding a new food source. Onlookers watch the dances of employed bees and choose food sources depending on dances. The main steps of the algorithm are given below.

```
Initial food sources are produced for all employed bees
REPEAT
    1. Each employed bee goes to a food source in her memory and determines a closest
       source, then evaluates its nectar amount and dances in the hive
    2. Each onlooker watches the dance of employed bees and chooses one of their sources
       depending on the dances, and then goes to that source. After choosing a neighbour
       around that, she evaluates its nectar amount.
    3. Abandoned food sources are determined and are replaced with the new food sources
       discovered by scouts.
    4. The best food source found so far is registered.
UNTIL (requirements are met)
```

Implementation for TSP

Food source: a path solution
Nectar amount: path length, the shorter the better
Neighbor food source: For any given solution, a neighbor is generated by picking any waypoint, generate a random number to replace it, then order the waypoints by smallest position values.
Best food source: shortest path

Performance Evaluation

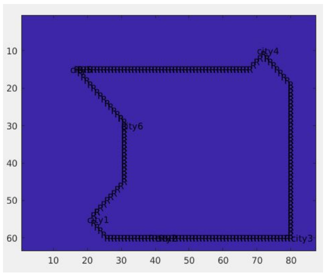
This is the shortest path generated for the given set of cities:

Num of Cities	Shortest path	Parameter 1: limit	Parameter 2: Max Cycle	Computing time(sec)
6	212	100	2.3K	0.03
10	237	220	240K	2.0
12	290	320	480K	5.8

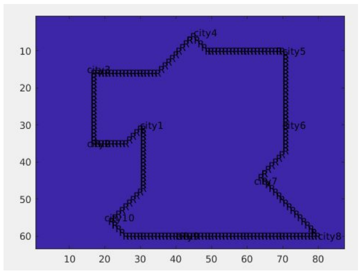
*Control parameter 1 <Limit>: A solution could not be improved through “limit” trials is abandoned by its employed bee

*Control parameter 2 <maxCycle>: The number of cycles for foraging

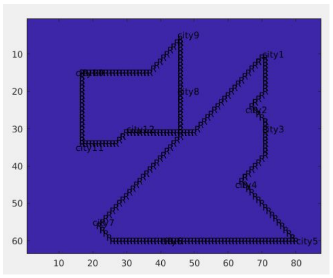
The result of 6 cities:



The result of 10 cities:



The result of 12 cities:


city9
city1
city8
city2
city3
city11
city4
city7
city5