File - /Users/jamieshepherd/Documents/ACO_BankSorting/ACO_API/API.c

```c
//
// Created by Jamie on 30/10/2019.
//

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "API.h"

/**
 * A function to print the current status of the network to console
 * @param network
 */
void printNetwork(Network network) {
    for (int nodeIndex = 0; nodeIndex < network.nodeCount; nodeIndex++) {
        printf("Node: %i\n", network.nodes[nodeIndex].nodeID);

        for (int pathIndex = 0; pathIndex < network.nodes[nodeIndex].pathCount; pathIndex++) {

            printf("Path: %i Destination node:%i Cost:%f Pheromone:%f\n",
                   pathIndex,
                   network.nodes[nodeIndex].paths[pathIndex].destinationNodeID,
                   network.nodes[nodeIndex].paths[pathIndex].cost,
                   network.nodes[nodeIndex].paths[pathIndex].pheromone
            );
        }
    }
}

/**
 * A function to simulate the ant moving through the network
 * @param antPointer
 * @param network
 */
void simulateAnt(Ant *antPointer, Network network) {
    while(1) {
        Node currentNode = network.nodes[antPointer->nodePath[antPointer->pathLength - 1]];

        int validPathCount = 0;
        Path *validPaths = malloc(validPathCount * sizeof(Path));

        for (int pathIndex = 0; pathIndex < currentNode.pathCount; pathIndex++) {

            bool alreadyVisited = false;
            for (int visitedPathIndex = 0; visitedPathIndex < antPointer->pathLength; visitedPathIndex++) {
                if (currentNode.paths[pathIndex].destinationNodeID == antPointer->nodePath[visitedPathIndex]) {
                    alreadyVisited = true;
                    break;
                }
            }

            if (!alreadyVisited) {
                validPathCount++;
                validPaths = realloc(validPaths, validPathCount * sizeof(Path));
                validPaths[validPathCount - 1] = currentNode.paths[pathIndex];
            }
        }

        if (validPathCount > 0) {
            antDecision(antPointer, validPaths, validPathCount);
        } else {
            break;
```

```
63        }
64
65
66        free(validPaths);
67      }
68
69  }
70
71  /**
72   * A function that probabilistically moves the ant along one of the given paths
73   * @param antPointer
74   * @param pathArray
75   * @param pathArraySize
76   */
77  void antDecision(Ant *antPointer, Path *pathArray, int pathArraySize) {
78      float totalPheromone = 0;
79
80      for (int pathIndex = 0; pathIndex < pathArraySize; pathIndex++) {
81          totalPheromone += pathArray[pathIndex].pheromone;
82      }
83
84      float randomValue = (rand() / (float) RAND_MAX) * totalPheromone;
85
86
87      float tempPheromoneValue = 0;
88      for (int pathIndex = 0; pathIndex < pathArraySize; pathIndex++) {
89          tempPheromoneValue += pathArray[pathIndex].pheromone;
90
91          if (tempPheromoneValue > randomValue) {
92              antPointer->pathLength++;
93              antPointer->nodePath = realloc(antPointer->nodePath, antPointer->pathLength * sizeof(int));
94              antPointer->nodePath[antPointer->pathLength - 1] = pathArray[pathIndex].destinationNodeID;
95
96              break;
97          }
98      }
99  }
100
101 /**
102  * A function that places pheromone in the network along the path of the given ant
103  * @param networkPointer
104  * @param ant
105  * @param pheromoneQuantity
106  */
107 void placePheromone(Network *networkPointer, Ant ant, float pheromoneQuantity) {
108     for (int antNodeIndex = 0; antNodeIndex < ant.pathLength - 1; antNodeIndex++) {
109         int currentNode = ant.nodePath[antNodeIndex];
110         int nextNode = ant.nodePath[antNodeIndex + 1];
111
112         for (int pathIndex = 0; pathIndex < networkPointer->nodes[currentNode].pathCount; pathIndex++) {
113             if (networkPointer->nodes[currentNode].paths[pathIndex].destinationNodeID == nextNode) {
114                 networkPointer->nodes[currentNode].paths[pathIndex].pheromone += pheromoneQuantity;
115                 break;
116             }
117         }
118     }
119 }
120
121 /**
122  * A function that modifies the pheromone throughout the network by a given scaling factor
123  * @param networkPointer
124  * @param evaporationRate
```

```
125  */
126 void evaporatePheromone(Network *networkPointer, float evaporationRate) {
127     for (int nodeIndex = 0; nodeIndex < networkPointer->nodeCount; nodeIndex++) {
128         for (int pathIndex = 0; pathIndex < networkPointer->nodes[nodeIndex].pathCount; pathIndex++) {
129             networkPointer->nodes[nodeIndex].paths[pathIndex].pheromone *= evaporationRate;
130         }
131     }
132 }
133
134
135
```

```
1  //
2  // Created by Jamie on 30/10/2019.
3  //
4
5  #ifndef ACO_BANKSORTING_API_H
6  #define ACO_BANKSORTING_API_H
7
8  typedef struct ant {
9      int pathLength;
10     int *nodePath;
11 } Ant;
12
13 typedef struct path {
14     int destinationNodeID;
15     float cost;
16     float pheromone;
17 } Path;
18
19 typedef struct node {
20     int nodeID;
21     int pathCount;
22     Path *paths;
23 } Node;
24
25 typedef struct network {
26     int nodeCount;
27     Node *nodes;
28 } Network;
29
30 /**
31  * A function to print the current status of the network to console
32  * @param network
33  */
34 void printNetwork(Network network);
35
36 /**
37  * A function to simulate the ant moving through the network
38  * @param antPointer
39  * @param network
40  */
41 void simulateAnt(Ant *antPointer, Network network);
42
43 /**
44  * A function that probabilistically moves the ant along one of the given paths
45  * @param antPointer
46  * @param pathArray
47  * @param pathArraySize
48  */
49 void antDecision(Ant *antPointer, Path *pathArray, int pathArraySize);
50
51 /**
52  * A function that places pheromone in the network along the path of the given ant
53  * @param networkPointer
54  * @param ant
55  * @param pheromoneQuantity
56  */
57 void placePheromone(Network *networkPointer, Ant ant, float pheromoneQuantity);
58
59 /**
60  * A function that modifies the pheromone throughout the network by a given scaling factor
61  * @param networkPointer
62  * @param evaporationRate
```

```
 */
void evaporatePheromone(Network *networkPointer, float evaporationRate);




#endif //ACO_BANKSORTING_API_H

```

```c
#include <stdio.h>
#include <time.h>
#include <math.h>
#include "ACO_API/API.h"
#include "bagAPI.h"


#if defined(__MACH__)
#include <stdlib.h>
#else
#include <malloc.h>
#endif


int main() {
    //Set the random seed based on current time (prevent same results every time)
    srand((unsigned int) time(0));

    //Load data file
    FILE *dataFile;
    dataFile = fopen("BankProblem.txt", "r");

    /*
     * File structure (repeating for n bags)
     *
     * 1  -security van capacity: %i
     * n+2 - bag %i:
     * n+3 -  weight: %f
     * n+4 -  value: %i
     *
     */

    //Setup data file variables
    int vanCapacity;
    int bagCount = 0;
    int bagID;
    Bag *bags = malloc(bagCount * sizeof(Bag));

    //Read data file contents
    fscanf(dataFile, "security van capacity: %i", &vanCapacity);

    while(fscanf(dataFile, " bag %i:", &bagID) != EOF) {
        bagCount++;
        bags = realloc(bags, bagCount * sizeof(Bag));
        fscanf(dataFile, "  weight: %f", &bags[bagCount - 1].weight);
        fscanf(dataFile, "  value: %i", &bags[bagCount - 1].value);
    }

    //Close data file
    fclose(dataFile);



    //Find fitness benchmark (compare packing efficiency of solution to average value to weight density)
    float totalWeight = 0;
    int totalValue = 0;
    for (int bagID = 0; bagID < bagCount; bagID++) {
        totalWeight += bags[bagID].weight;
        totalValue += bags[bagID].value;
    }
    float averageValueDensity = totalValue / totalWeight;
    float estimatedValue = averageValueDensity * vanCapacity;
```

```
63
64
65
66
67
68  int testsToRun = 10; //Number of tests to run for each setup
69  int fitnessEvaluations = 10000; //Number of fitness evaluations to use per test
70
71
72  int populationSizeSet[5] = {5, 10, 25, 50, 100}; //Population sizes to test
73
74  for (int populationSizeIndex = 0; populationSizeIndex < 5; populationSizeIndex++ ){
75    int p = populationSizeSet[populationSizeIndex];
76
77    for (float e = 0.5; e <= 0.9; e+=0.1) { //Evaporation rate
78
79      for (float m = 0.0001; m < 1; m*=10) { //Amount of pheromone to place based on fitness
80
81        //Accumulate attributes of best ants to find mean of all tests
82        float cumulativeBestValues = 0;
83        float cumulativeBestWeights = 0;
84
85        for (int test = 0; test < testsToRun; test++) {
86          //Initialise network
87          Network network;
88          network.nodeCount = (bagCount * 2) + 2; //In or out stats for bag plus start and end node
89          network.nodes = malloc(network.nodeCount * sizeof(Node));
90
91          for (int nodeIndex = 0; nodeIndex < network.nodeCount; nodeIndex++) {
92            /*
93             * First and last node considered as start and end respectively
94             * All other nodes come in pairs representing the two states (in van / not in van) of the bag
95             * All nodes should be connected ONLY to bags not yet sorted (higher bag index)
96             *
97             * Bag index given by: ((nodeIndex + nodeIndex % 2) / 2) - 1
98             * Note this marks start and end nodes with bag indexes however this is to be ignored
99             */
100
101            network.nodes[nodeIndex].nodeID = nodeIndex;
102
103            if (nodeIndex == 0) {
104              //START node - To in or out state of first bag
105              network.nodes[nodeIndex].pathCount = 2;
106            } else if (nodeIndex == network.nodeCount - 1) {
107              //END node - Can't go anywhere
108              network.nodes[nodeIndex].pathCount = 0;
109            } else if ((nodeIndex + nodeIndex % 2) / 2 == bagCount) {
110              //Nodes for last bag - To end node
111              network.nodes[nodeIndex].pathCount = 1;
112            } else {
113              //Nodes for all but last bags - To either in or out state of next bag
114              network.nodes[nodeIndex].pathCount = 2;
115            }
116
117            network.nodes[nodeIndex].paths = malloc(network.nodes[nodeIndex].pathCount * sizeof(Path));
118
119            for (int pathIndex = 0; pathIndex < network.nodes[nodeIndex].pathCount; pathIndex++) {
120              network.nodes[nodeIndex].paths[pathIndex].destinationNodeID =
121                  nodeIndex + nodeIndex % 2 + (pathIndex + 1);
122              network.nodes[nodeIndex].paths[pathIndex].cost = 1;
123              network.nodes[nodeIndex].paths[pathIndex].pheromone = rand() / (float) RAND_MAX;
124            }
```

```
125             }
126
127
128             //Run ant simulation
129
130             //Initialise location to save best ant
131             Ant bestAnt;
132             bestAnt.pathLength = 1;
133             bestAnt.nodePath = malloc(1 * sizeof(int));
134
135             //Save metrics for best ant
136             float bestFitness = 0;
137             float bestAntWeight = 0;
138             int bestAntValue = 0;
139
140             //Iterate through the generations until reaching end condition of X fitness evaluations
141             for (int generation = 0; generation < fitnessEvaluations / p; generation++) {
142               Ant *ants = malloc(p * sizeof(Ant));
143               float *fitnesses = malloc(p * sizeof(float));
144
145               //Setup generation
146               for (int antIndex = 0; antIndex < p; antIndex++) {
147                 ants[antIndex].pathLength = 1;
148                 ants[antIndex].nodePath = malloc(ants[antIndex].pathLength * sizeof(int));
149                 ants[antIndex].nodePath[0] = 0;
150
151                 //Simulate ant movement
152                 simulateAnt(&ants[antIndex], network);
153
154
155
156                 // Calculate fitness
157                 float loadedWeight = 0;
158                 int loadedValue = 0;
159                 int loadedBags = 0;
160                 for (int bagID = 0; bagID < bagCount; bagID++) {
161                   if (ants[antIndex].nodePath[bagID] % 2 == 0) {
162                     loadedWeight += bags[bagID].weight;
163                     loadedValue += bags[bagID].value;
164                     loadedBags++;
165                   }
166                 }
167
168                 fitnesses[antIndex] = (loadedValue / estimatedValue);
169
170                 if (loadedWeight > vanCapacity) {
171                   fitnesses[antIndex] *= vanCapacity / totalWeight;
172                 }
173
174                 fitnesses[antIndex] = pow(fitnesses[antIndex], 8);
175
176
177
178
179                 //Save this as the best ant if relevant
180                 if (fitnesses[antIndex] > bestFitness) {
181                   bestAnt.pathLength = ants[antIndex].pathLength;
182                   bestAnt.nodePath = realloc(bestAnt.nodePath, bestAnt.pathLength * sizeof(int));
183                   for (int i = 0; i < ants[antIndex].pathLength; i++) {
184                     bestAnt.nodePath[i] = ants[antIndex].nodePath[i];
185                   }
186
```

```
187                     bestFitness = fitnesses[antIndex];
188                     bestAntWeight = loadedWeight;
189                     bestAntValue = loadedValue;
190                   }
191                 }
192
193
194                 for (int antIndex = 0; antIndex < p; antIndex++) {
195                   //Increase pheromone
196                   placePheromone(&network, ants[antIndex], m * fitnesses[antIndex]);
197                   free(ants[antIndex].nodePath);
198                 }
199                 free(ants);
200                 free(fitnesses);
201
202
203
204                 //Evaporate pheromone
205                 evaporatePheromone(&network, e);
206               }
207
208               //Increment cumulative best values and weights
209               cumulativeBestValues += bestAntValue;
210               cumulativeBestWeights += bestAntWeight;
211             }
212
213
214             printf("p:%i \t e:%g \t m:%g \t Mean value packed: %g \t Mean weight packed: %g \n", p, e, m,
    cumulativeBestValues / testsToRun, cumulativeBestWeights / testsToRun);
215           }
216           printf("\n");
217         }
218         printf("\n\n");
219       }
220
221       return 0;
222   }
```

File - /Users/jamieshepherd/Documents/ACO_BankSorting/CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.10)
2  project(ACO_BankSorting C)
3
4  set(CMAKE_C_STANDARD 99)
5
6  set(SOURCE_FILES main.c ACO_API/API.c ACO_API/API.h bagAPI.h)
7
8  add_executable(ACO_BankSorting ${SOURCE_FILES})
```

```
1  //
2  // Created by Jamie on 30/10/2019.
3  //
4
5  #ifndef ACO_BANKSORTING_BAGAPI_H
6  #define ACO_BANKSORTING_BAGAPI_H
7
8  typedef struct bag{
9      float weight;
10     int value;
11 } Bag;
12
13 #endif //ACO_BANKSORTING_BAGAPI_H
14
```