Faculty of Engineering and Technology

Computer Science Department

Artificial Intelligence (COMP338)

Assignment 2

Seating Arrangement Optimization

Name: Sama Wahidee

ID: 1211503

Section: 1

Instructor: Radi Jarrar

Date: 10/06/2024

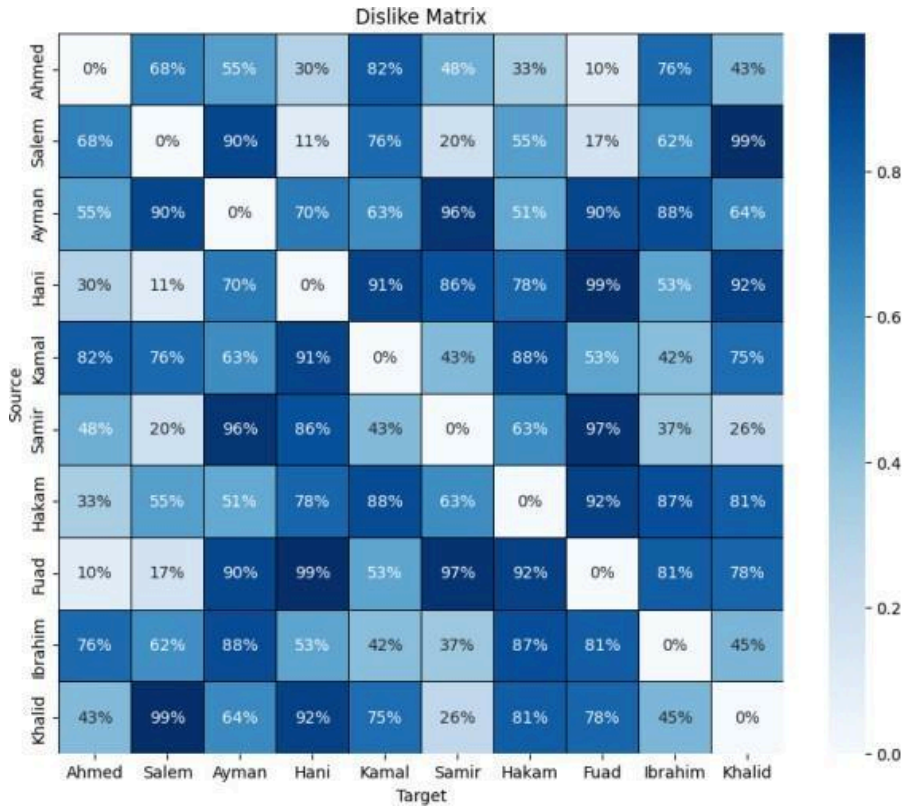
Contents:

Introduction (Problem Description):

As for the second Project , the problem of how to arrange seating when placing a round table is discussed in this report. The idea is to avoid having situations that some people may feel awkward and which can cause conflicts among participants. There is also a “dislike table” given to enumerate the overall amount of negative contact between two people. It should be noted that this table is used as a heuristics to help with solving the problems related to the seating decisions.

The problem of designing such a network is compounded by the fact that in addition to connecting the participants to all other participants, each participant must also communicate with the participants in the closest physical proximity. Thus, the idea of the set up of the classroom chairs and tables should be such that it should reduce conflict and at the same time encourage communication as well as inspire the corp among the group.

This proposed report shall solely aim to find out which algorithm is best suited to seat the attendees. It also should maintain a non-aggressive approach according to the dislike table and non-linear dislike cost function.



Algorithms Implementation:

- The code was built on the Eclipse environment using Java

This investigation employed three optimization algorithms: There are three forms of optimization that can be associated with the problem and they include Genetic Algorithm, Simulated Annealing, and Hill Climbing. The purpose of such algorithms was to look for the correct positioning of the people around a round table. Ideally, they should arrange themselves in a way that achieves the least total dislike cost, as suggested in the following 10x10 dislike matrix. Every algorithm tries a number of possibilities as to form the best seating arrangement with figures are for round tables.

1. Genetic Algorithms:

Derived from the principles of evolutionary biology and mechanisms of natural selection, GAs employ similar ideas like inheritance, mutation, sorting, and crossover in the process of optimization and search. GAs are classified under a group of algorithms known as global search heuristics, which can be used in the evolution of the solution to higher problems through the improvement of the population of potential solutions.

Code Explanation:

1. **Initialization**:A population of random seating arrangements is generated by shuffling the list of guests.

```
for (int i = 0; i < populationSize; i++) { List<String>
arrangement = new ArrayList<>(Arrays.asList(guests));
Collections.shuffle(arrangement); population.add(arrangement); }
```

2. **Fitness Evaluation**:The population is sorted based on the cost of each arrangement, calculated by the `calculateCost` function.

```
population.sort(Comparator.comparingDouble(arr ->
calculateCost(arr, graph)));
```

3. **Selection**:The top 60% of the population (with the lowest costs) are selected to be parents for the next generation.

```
int cutoff = (int) (0.6 * populationSize);
List<List<String>> newPopulation = new
ArrayList<>(population.subList(0, cutoff));
```

4. **Crossover**
    - Pairs of parents are randomly selected, and the `crossover` function is used to create new offspring arrangements. The `crossover` function combines parts of both parents.

```
while (newPopulation.size() < populationSize) {
    List<String> parent1 =
population.get(random.nextInt(cutoff));
    List<String> parent2 =
population.get(random.nextInt(cutoff));
    List<String> child = crossover(parent1, parent2);
    mutate(child, mutationRate);
    newPopulation.add(child);
}
```

    - A random crossover point is chosen, and the child starts with the first part from one parent and completes with non-duplicated elements from the other parent.

```
static List<String> crossover(List<String> parent1, List<String>
parent2) {
    int crossoverPoint = random.nextInt(parent1.size());
    Set<String> childSet = new HashSet<>(parent1.subList(0,
crossoverPoint));
    List<String> child = new ArrayList<>(parent1.subList(0,
crossoverPoint));
    for (String guest : parent2) {
        if (!childSet.contains(guest)) {
            child.add(guest);
            childSet.add(guest);
        }
    }
    return child;
}
```

5. **Mutation**:With a certain mutation rate, random swaps are performed in the arrangement to introduce genetic diversity.

```
static void mutate(List<String> arrangement, double mutationRate)
{
    if (random.nextDouble() < mutationRate) {
        int index1 = random.nextInt(arrangement.size());
        int index2 = random.nextInt(arrangement.size());
        Collections.swap(arrangement, index1, index2);
    }
}
```

6. **Replacement:** The new generation replaces the old population, and the process is repeated for a specified number of generations.

2. Simulated Annealing:

Simulated Annealing is a stochastic optimization technique developed rudimentarily from the metallurgical simulation of annealing, which involves heating and controlled cooling of metals to enhance their stability in terms of crystal structure. Concurrently, in optimization it searches through the solution space by allowing 'incorrect' moves first (high temperature), but in the longer run, increases the likelihood of 'correcting' the suboptimal solution (low temperature) to avoid ending up at local optima.

Code Explanation:

1. **Initialization**:The algorithm starts by initializing a random arrangement of guests and setting the initial temperature.

```
List<String> currentArrangement = new
ArrayList<>(Arrays.asList(guests));
Collections.shuffle(currentArrangement); List<String>
bestArrangement = new ArrayList<>(currentArrangement);
double currentCost = calculateCost(currentArrangement, graph);
double bestCost = currentCost;
double temperature = initialTemperature;
for (int i = 0; i < numIterations; i++) {
List<String> newArrangement = newArrayList<>(currentArrangement);
int index1 = random.nextInt(newArrangement.size()); int index2 =
random.nextInt(newArrangement.size());
Collections.swap(newArrangement, index1, index2);
 double newCost = calculateCost(newArrangement, graph);
```

2. **Acceptance Probability Calculation:**Calculate the probability of accepting the new arrangement based on the cost difference and current temperature.

```
if (acceptanceProbability(currentCost, newCost,temperature) >
random.nextDouble()) {
            currentArrangement = newArrangement;
            currentCost = newCost;
        }
```

3. **Acceptance Criteria:** Accept the new arrangement if it has a lower cost. If it has a higher cost, accept it with a probability determined by the acceptance probability function.

```
    if (newCost < bestCost) {
     bestArrangement = newArrangement;
     bestCost = newCost;
    }
   temperature *= coolingRate;
  }
  return bestArrangement;
}
```

4. **Acceptance Probability Function**: This function calculates the probability of accepting a new arrangement based on the current cost, new cost, and current temperature.

```
static double acceptanceProbability(double currentCost, double
newCost, double temperature) {
    if (newCost < currentCost) {
        return 1.0;
    }
    return Math.exp((currentCost - newCost) / temperature);
}
```

5. **Cost Calculation:**This function computes the total cost of a given seating arrangement based on the dislike values between adjacent guests.

```
static double calculateCost(List<String> arrangement, Graph
graph) {
    double totalCost = 0;
    for (int i = 0; i < arrangement.size() - 1; i++) {
        String guest1 = arrangement.get(i);
        String guest2 = arrangement.get(i + 1);
        int index1 = Arrays.asList(guests).indexOf(guest1);
        int index2 = Arrays.asList(guests).indexOf(guest2);
        totalCost += graph.adjMatrix[index1][index2];
    }
int lastIndex =
Arrays.asList(guests).indexOf(arrangement.get(arrangement.size()
- 1));
    int firstIndex =
Arrays.asList(guests).indexOf(arrangement.get(0));
    totalCost += graph.adjMatrix[lastIndex][firstIndex];
    return totalCost;
}
```

3. Hill Climbing:

Hill Climbing is a local search algorithm which picks a neighboring solution from the current solution on its search space then updates its best solution currently stored and stores the new solution if its cost is lower than current best one. You have incorporated the Hill Climbing algorithm in your program and use it to place the guests according to the dislike percentages.

Code Explanation:

1. **Initialization**:e algorithm starts by initializing a random arrangement of guests and calculating its cost.This process is repeated for a specified number of restarts to avoid local minima.

```
static List<String> hillClimbing(Graph graph, int numRestarts) {
    List<String> bestArrangement = new
ArrayList<>(Arrays.asList(guests));
    Collections.shuffle(bestArrangement);
    double bestCost = calculateCost(bestArrangement, graph);
```

2. **Multiple Restarts:**The algorithm performs a certain number of restarts to explore different starting points in the solution space.

```
for (int restart = 0; restart < numRestarts; restart++) {
        List<String> currentArrangement = new
ArrayList<>(Arrays.asList(guests));
        Collections.shuffle(currentArrangement);
        double currentCost = calculateCost(currentArrangement,
graph);
```

3. **Local Search for Improvement:**The algorithm enters a loop where it tries to improve the current arrangement by swapping pairs of guests.If a swap results in a lower cost, the new arrangement is accepted as the current arrangement, and the search continues from there.

```
    boolean improved;
        do {
            improved = false;
            for (int i = 0; i < currentArrangement.size() - 1;
i++) {
                for (int j = i + 1; j <
currentArrangement.size(); j++) {
                    List<String> newArrangement = new
ArrayList<>(currentArrangement);
                    Collections.swap(newArrangement, i, j);
                    double newCost =
calculateCost(newArrangement, graph);
                    if (newCost < currentCost) {
                        currentArrangement = newArrangement;
                        currentCost = newCost;
                        improved = true;
                    }
                }
            }
        } while (improved);
```

4. **Update Best Arrangement:**After each restart, if the current arrangement found during the local search has a lower cost than the best known arrangement, it updates the best arrangement.

```
      if (currentCost < bestCost) {
            bestArrangement = currentArrangement;
            bestCost = currentCost;
        }
    }

    return bestArrangement;
}
```

Used variables in each algorithm:

1. Genetic Algorithms:

- **populationSize:** A count of all the people in the society or target population of a given society or country.it used to determines the number of states or nodes utilized in the algorithm.It's value 100

- **numGenerations:** The total generations that will be used to solve the problem with the genetic algorithm.it used to Specify whether the population evolves once or more than once, as well as how many times.It's value 1000

- **mutationRate:** This is the likelihood that an individual shall undergo a mutation.it used to serve to bring genetic variation to counteract the problem of greedy optimization which easily converges.It's value 0. 1

- **population:** A list of the current population of entire participants and their seating arrangements.It used to Store the current population being tested .

- **cutoff:** The genotype of the number of persons involved in production.It used to impose restriction or limitation to the selection pressure by defining the number of organisms to be retained in the next generation.

- **newPopulation:** New population that is produced by selection, crossover and mutation of the current population.It used to hold the potential of storing the offspring for the next generation.

- **crossover:** The technique utilized in breeding of two parent individuals to develop progeny.It used genetic information from parents is combined to develop new solutions.

- **mutate:** It refers to the procedure used to make the mentioned mutations in an individual.It used to publish two guests and exchange their positions randomly with a certain probability set to the mutation rate.

2. Simulated Annealing:

- **initialTemperature:** The starting temperature of the process of increasing the temperature to a desired level.It used to settle the control of the first acceptance probability of worse solutions control.It's value 1000

- **coolingRate:** The extent to which the temperature reduces when it is exposed to cool or a cold region is referred to as the rate of cooling.It used to influence the rate at which the temperature decreases, which defines the flexibility of the solution search space.It's value 0. 99

- **numIterations:** The number of iterations for the simulated annealing process as couples are matched.It is used to determine the length of the algorithm when the output exceeds the specified value.It's value 10000

- **currentArrangement:** The existing configuration of students currently seated in random rows.It used to preserve the existing solution within the process.

- **bestArrangement:** Even better is the left side of the back row of a large lecture hall or the back row of middle sections between aisles of an large auditorium.It used to save the solution which is unique and best among all the other possible solutions in problem solving.

- **temperature:** The current setting for the process where the material is exposed to a specific temperature for a certain period.It used to shift the likelihood of accepting worse solutions in the course of the bumping making it more, less or equal to that of accepting worse solutions before the bumping.

3. Hill Climbing:

- **numRestarts**: The number of times the hill climbing process is restarted.It used to help avoid local minima by restarting the process multiple times.It's value 100.

- **currentArrangement**: The current seating arrangement being evaluated.It used to maintain the current solution in the process.

- **bestArrangement**: The best seating arrangement found so far.It used to store the optimal solution discovered during the process.

- **improved**: A boolean flag indicating whether an improvement has been made in the current iteration.It used to control the continuation of the local search loop.

Results:

After implementing Genetic Algorithm, Simulated Annealing, and Hill Climbing for the Round Table Seating Arrangement problem, we obtained the following results:

First Run

- ### Genetic Algorithms:

Seating Arrangement:[Hakem, Ahmad, Fuad, Salem, Hani, Ayman, Kamal, Ibrahim, Khalid, Samir]

Total Cost: 3.8

```
Genetic Algorithm Seating Arrangement: [Hakem, Ahmad, Fuad, Salem, Hani, Ayman, Kamal, Ibrahim, Khalid, Samir]
Total Cost: 3.8
```

- ### Simulated Annealing:

Seating Arrangement: [Khalid, Samir, Salem, Fuad, Ahmad, Hakem, Ayman, Hani, Ibrahim, Kamal]

Total Cost: 3.9699999999999998

```
Simulated Annealing Seating Arrangement: [Khalid, Samir, Salem, Fuad, Ahmad, Hakem, Ayman, Hani, Ibra
Total Cost: 3.9699999999999998
```

- ### Hill Climbing:

Seating Arrangement:[Fuad, Ahmad, Hakem, Ayman, Khalid, Samir, Kamal, Ibrahim, Hani, Salem]

Total Cost: 3.4999999999999996

```
Hill Climbing Seating Arrangement: [Fuad, Ahmad, Hakem, Ayman, Khalid, Samir, Kamal, Ibrahim, Hani, Salem]
Total Cost: 3.4999999999999996
```

Second Run

- ## Genetic Algorithms:

Seating Arrangement:[Ibrahim, Kamal, Ayman, Hakem, Hani, Salem, Fuad, Ahmad, Khalid, Samir]

Total Cost: 3.7800000000000002

```
Genetic Algorithm Seating Arrangement: [Ibrahim, Kamal, Ayman, Hakem, Hani, Salem, Fuad, Ahmad, Khalid, Samir]
Total Cost: 3.7800000000000002
```

- ## Simulated Annealing:

Seating Arrangement: [Fuad, Salem, Hani, Ibrahim, Kamal, Samir, Khalid, Ayman, Hakem, Ahmad]

Total Cost: 3.5000000000

```
Simulated Annealing Seating Arrangement: [Fuad, Salem, Hani, Ibrahim, Kamal, Samir, Khalid, Ayman, Hakem, Ahmad]
Total Cost: 3.5000000000000004
```

000004

- ## Hill Climbing:

Seating Arrangement:[Fuad, Ahmad, Hakem, Ayman, Khalid, Samir, Kamal, Ibrahim, Hani, Salem]

Total Cost: 3.4999999999999996

```
Hill Climbing Seating Arrangement: [Fuad, Ahmad, Hakem, Ayman, Khalid, Samir, Kamal, Ibrahim, Hani, Salem]
Total Cost: 3.4999999999999996
```

Third Run

- ## Genetic Algorithms:

Seating Arrangement:[Hakem, Ahmad, Fuad, Kamal, Ibrahim, Khalid, Samir, Salem, Hani, Ayman]

Total Cost: 3.6099999999999994

```
Genetic Algorithm Seating Arrangement: [Hakem, Ahmad, Fuad, Kamal, Ibrahim, Khalid, Samir, Salem, Hani, Ayman]
Total Cost: 3.6099999999999994
```

- ## Simulated Annealing:

Seating Arrangement: [Salem, Hakem, Ayman, Khalid, Samir, Kamal, Ibrahim, Hani, Ahmad, Fuad]

Total Cost:3.9099999999999997

```
Simulated Annealing Seating Arrangement: [Salem, Hakem, Ayman, Khalid, Samir, Kamal, Ibrahim, Hani, Ahmad, Fuad]
Total Cost: 3.9099999999999997
```

- ## Hill Climbing:

Seating Arrangement:[Kamal, Samir, Khalid, Ayman, Hakem, Ahmad, Fuad, Salem, Hani, Ibrahim]

Total Cost: 3.5

```
Hill Climbing Seating Arrangement: [Kamal, Samir, Khalid, Ayman, Hakem, Ahmad, Fuad, Salem, Hani, Ibrahim]
Total Cost: 3.5
```

Explain and compare the results

1. Genetic Algorithms

- **Explanation:**

GA unveils the solution space as a set of potential solution constructs and recombination them using selection, crossover, and mutation across generations. In each generation, a number of points are selected for reproduction according to the values of the fitness function; in other words, given higher fitness values, not only are better solutions produced, but the regions of decision space containing these solutions are explored as well. Although, one disadvantage of crossover and mutation operations is that these are probabilistic in nature, meaning that they can create very low-quality solutions at times.

- **Comparison of Results:**

It is also noteworthy that the obtained score in GA fluctuates from run to run as GA is defined as the stochastic algorithm. In the first run, the total cost with regards not only to the location but to the setting as well was equal to 3. Perhaps, that is why the obtained value of the eighth variable, which is 8, looks quite satisfactory and corresponds to a relatively good solution. However in the successive iterations, it swayed through out Total costs rationally varying between 3. 5 to 3. 78. This becomes significantly higher to indicate that GA may need more iterations or the population size to be large enough to yield good results regularly.

- **Justification:**

GA's capability of expanding over a vast solution space and the interactive usage of various solutions make it possible for the algorithm to find good solutions as noted by the lowest total cost as found in some runs. However the variances in the performance point towards the dependence on factors including population size, crossover rate and the mutation rate. In situations that require higher accuracy, these parameters may need to be changed or their number increased in order to achieve the appropriate level of output sample homogeneity.

2. Simulated Annealing:

- **Explanation:**

SA is quite similar to the actual annealing process whereby an item is slowly cooled down to a lower energy state. Likewise, SA uses an initial solution and carries out search in the neighborhood of the solution and accepts worse solutions according to the probability controlled by temperature parameter descent over time. This also enables SA to get out of local optima and starting searching for the solutions globally in the solution space.

- **Comparison of Results:**

It has been shown that SA generates high variability across different runs meaning that is not very accurate. Thus in some runs it found ‘‘good’’ solutions with total costs in the grip of 3. 5, while in other cases the total costs overrun the figure of 3. 9, indicating suboptimal solutions. This variability (i.e., large variance) points toward the fact that SA seems to be sensitive to parameter settings and the cooling schedule used.

- **Justification:**

The opportunity to escape the local optimum can be regarded as one of the primary advantages of SA and make it search a considerably wider region of the solution space in contrast to pure deterministic algorithms such as for example hill climbing. However, the reaction is dependent on some factors like: initial temperature, and cooling rate. The case with SA can be said to have been best when SA performed well, whereby it gets solutions close towards the actual global optima. On the other hand, in runs with suboptimal solutions some states may be cooled such that there was little exploration of the space by the cooling schedule.

3. Hill Climbing:

- **Explanation:**

HC keeps on changing the current solution slightly to come up with a better solution. It is predictive in nature and always takes the best neighboring solution hence best for searching for local optimums. However, it depends highly on the initialization of the variables and can end up at some local minima, not the global minimum.

- **Comparison of Results:**

HC was seen to solve problems well and attain total cost of 3 to 4 across the runs as depicted below. 5 to 3. 8. Although it may not always reach the best solution it seeks, the use of deterministic tool is guaranteed and has about it. But, it lets optima go while it searches, and may not find the global optima especially in complicated problem spaces.

- **Justification:**

By studying the above table, one would realize that HC is very reliable for coming up with good solutions since its solution quality will not change over different runs. The fact that it is simple and deterministic optimal makes it appropriate for situations where getting the local optimums will suffice. However it does lose points in its ability to further a global search through the solution space due to its weakness to getting stuck in local optima.

Overall Justification and Comparison

1. Genetic Algorithms:

- **Performance:** Varies across runs but shows potential for finding good solutions.
- **Strengths:** Can explore a wide solution space and combine different solutions.
- **Weaknesses:** Can be inconsistent and sometimes produce suboptimal solutions.
- **Overall:** Effective but may require more generations or population size for better consistency.

2. Simulated Annealing:

- **Performance:** Highly variable, with both good and suboptimal results.
- **Strengths:** Good at escaping local minima and exploring the solution space.
- **Weaknesses:** Performance depends heavily on cooling schedule parameters.
- **Overall:** Can be very effective but is highly sensitive to parameter settings.

3. Hill Climbing:

- **Performance:** Consistently finds good solutions with low variability.
- **Strengths:** Simple and reliable for finding local optima.
- **Weaknesses:** Prone to getting stuck in local minima.
- **Overall:** Very reliable for consistent performance but may miss global optima in more complex problems.

Challenges i have faced

1. **Handling Circular Seating:**
   The basic optimization algorithms could be the issue of applying some circular configurations directly.
   I have, for instance, had to introduce modifications to the algorithms where this means that the first and last people in the list are also included.

2. **Parameter Tuning:**
   These algorithms are achieved with set parameters such as population size, mutation rate, temperature, etc. that can be optimized in terms of performance.
   Going through evaluations is crucial to optimizing the successes experienced.

3. **Scalability:**
   The idea might not be very efficient when dealing with a larger number of people since the number of possible arrangements is much larger and the computation time required to find a solution is increased significantly.

4. **Determine results:**
   The problem made it hard to judge the credibility of the solution and the results because of the complexity involved in arithmetical computation.

References:

1. COMP338.6.LocalSearch.Fall2024
2. Difference Between Hill Climbing and Simulated Annealing Algorithm