Faculty of Engineering and Information Technology

Computer Science Department

Artificial Intelligence

COMP338 : Assignment #1

Prepared by:

Worood Assi – 1210412

Instructor:

Radi Jarrar

Date :

21-4-2024

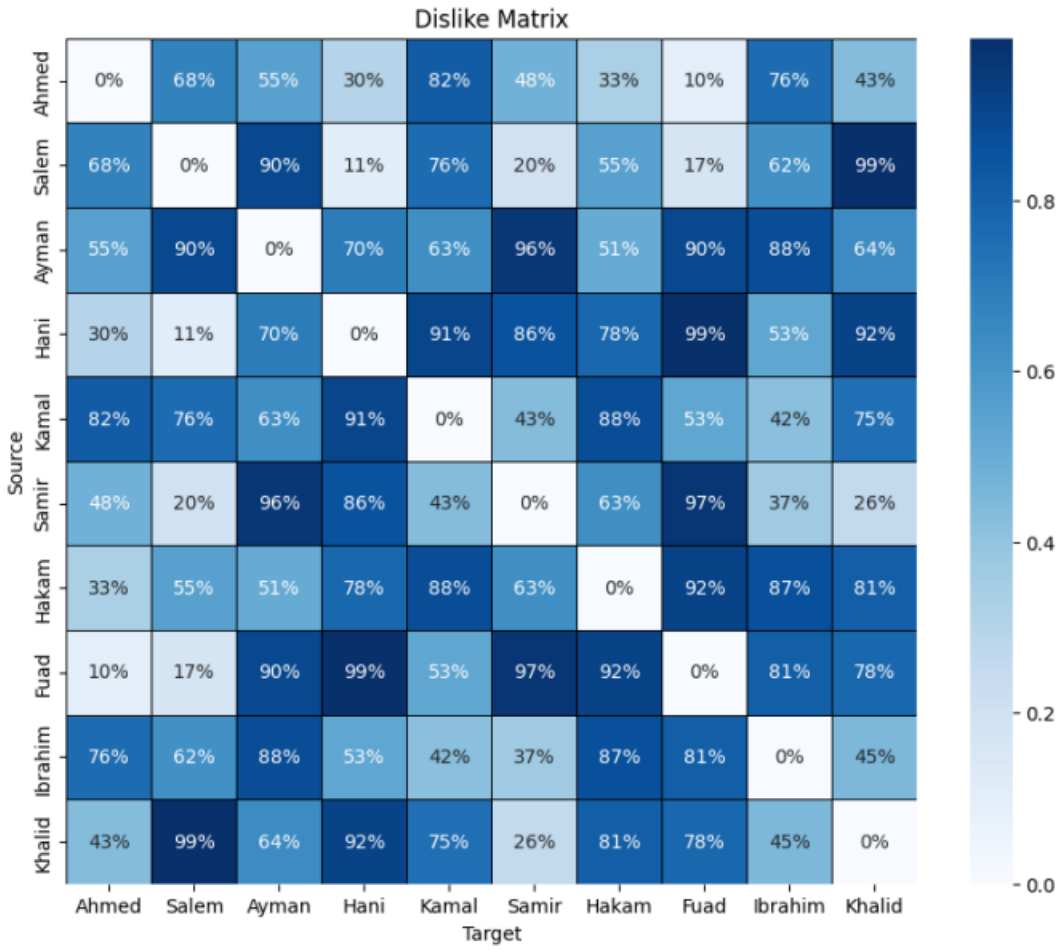
Contents

Problem:

Round Table Seating Arrangement

Clarification:

Based on the following inferred table that represents the percentages of dislikes between pairs of individuals, indicating the level of conflict or discomfort that each person feels towards the others. The goal is to seat individuals around a round table in the most appropriate arrangement to minimize conflict while ensuring that each person can talk to their neighbors on the left and right.



Heuristic Table (Dislike Matrix):

The heuristic table provides dislike percentages between pairs of individuals, with higher percentages indicating greater dislike or conflict. For example, a dislike percentage of 20% between Person A and Person B indicates a low level of conflict, while a dislike percentage of 80% indicates a high level of conflict.

Objective:

The objective is to determine which algorithm among Uniform Cost Search (UCS), Greedy Search, and A* Search can find the best seating arrangement that minimizes conflict based on the provided heuristic table and the Non-Linear Dislike Cost function.

Greedy Best First Search:

Greedy Search is an uninformed search algorithm that prioritizes nodes closest to the goal state, aiming to reach the solution quickly. It evaluates nodes solely based on a heuristic function, disregarding actual edge weights in a weighted graph. The algorithm expands nodes that seem closest to the goal, iteratively building a tree starting from the source node. It maintains an open list of nodes to explore, selecting the node with the lowest h(x) value for expansion. If a child node reaches the target, the algorithm succeeds; otherwise, it continues exploring until finding a solution or exhaustively searching the space.

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

Uniform Cost Search (UCS):

Uniform-Cost Search (UCS) is a variant of Dijkstra's algorithm tailored for weighted search graphs, prioritizing the exploration of nodes based on the lowest total cost from the root to the current node. Unlike breadth-first search (BFS), which prioritizes shallow exploration, UCS aims to find the cheapest solution. By evaluating the cost for each reachable node and expanding the node with the lowest path cost, UCS ensures efficiency. This process involves maintaining a priority queue organized by path cost. While the algorithm largely resembles the general graph search algorithm, it incorporates the priority queue mechanism and includes an extra check to handle the discovery of a shorter path to a frontier state.

Uniform-cost search - Algorithm

```
function UNIFORM-COST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

A* search:

A* Search is a variant of Best-First Search that evaluates nodes based on a combination of the cost to reach the node (g(n)) and the heuristic cost to get from the node to the goal (h(n)), represented by the evaluation function f(n) = g(n) + h(n). It is like the Dijkstra's algorithm but diverges in the choice of the evaluation function for determining the next node to explore. At each step, A* expands only the nodes with the lowest value of f(n), terminating when the goal node is found. It is both complete and optimal, offering a reasonable approach to finding the cheapest solution by prioritizing nodes with the lowest combined cost. A* search shares similarities with Uniform-Cost Search but uses g + h instead of just g.

Algorithm 24 A^* Algorithm

Input: A graph

Output: A path between start and goal nodes

repeat
  Pick n_{best} from O such that $f(n_{best}) \leq f(n), \forall n \in O$.
  Remove n_{best} from O and add to C.
  If $n_{best} = q_{goal}$, EXIT.
  Expand n_{best}: for all $x \in \text{Star}(n_{best})$ that are not in C.
  if $x \notin O$ **then**
    add x to O.
  else if $g(n_{best}) + c(n_{best}, x) < g(x)$ **then**
    update x's backpointer to point to n_{best}
  end if
until O is empty

In this project, the values in the dislike table represent h(n) for greedy search, the square of this value represents g(n) for UCS, and f(n) = h(n) + g(n) for A* search.

Evaluating the effectiveness of algorithms:

Greedy Algo.

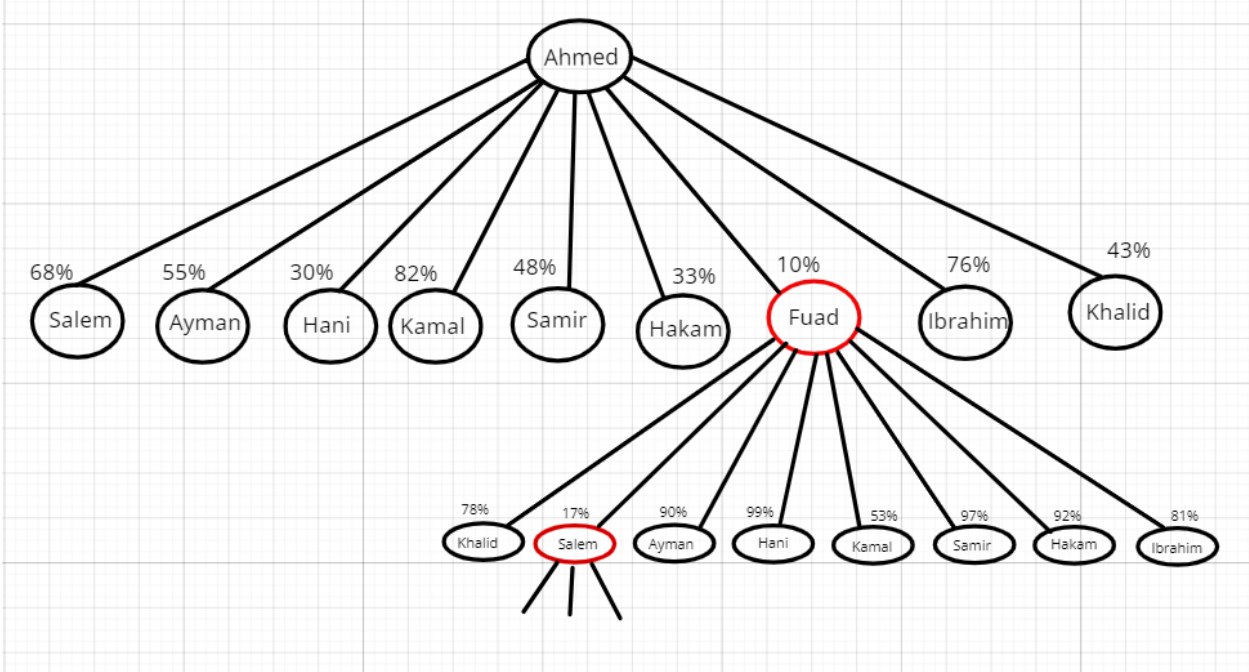
To calculate the lowest seating cost using the greedy algorithm, the lowest cost for each node was calculated linearly without relying on previous values, and when starting the search from Ahmed, the result is like this,

```
Greedy =>

TOTAL COST => 4.389999933540821
[Ahmed, Fuad, Salem, Hani, Ibrahim, Samir, Khalid, Ayman, Hakam, Kamal]
```

The person who dislikes Ahmed the least and sits next to him is Fouad, then the person who dislikes Fouad the least is Hani, and so on. The percentage of hatred for each person is added up.

It is important when choosing a person that he has not been seated before.



UCS Algo.

Using UCS algorithm, the lowest cost was calculated for each node by squaring the dislike rate with the other person. The search was based on the previous values that were calculated to find the lowest cost pass and not the shortest, such as greedy, and when starting the search from Ahmed, After calculating the cost for each person, Fouad is the lowest, then the cost between Fouad and the rest of the people is calculated taking the first cost into account, and so on.

It is important when choosing a person that he has not been seated before.

The final result is:

```
UCS =>

TOTAL COST => 1.5393999412417427
[Ahmed, Fuad, Salem, Hani, Ibrahim, Kamal, Samir, Khalid, Ayman, Hakam]
```

A* Algo.

The A* algorithm relies on the cost to reach node n from the starting state and the heuristic value to calculate the cost per node. The dislike rate is summed with the real cost to reach this node and then the lowest cost is selected, starting the search from Ahmed, so that the person with the lowest dislike rate is selected and sits next to Ahmed, and so on. The result is like this,

```
A* =>

TOTAL COST => 3.409999946653843
[Ahmed, Fuad, Salem, Hani, Ibrahim, Kamal, Samir, Khalid, Ayman, Hakam]
```

Summary

- The greedy algorithm tends to choose the option that seems best at the moment without considering the potential long-term consequences. While it's quick and often finds a solution, it doesn't always pick the most cost-effective path overall.

- The UCS algorithm prioritizes options based on their current costs while also considering their potential long-term impacts. It frequently discovers solutions and consistently selects the most cost-effective path overall. Its primary focus is on identifying the lowest-cost solution rather than simply finding the shortest route.

    - When reaching Ibrahim, Kamal was chosen to sit next to him and not Samir as the greedy one because it represents the lowest cost

- Regarding the A* algorithm, it achieved a similar ranking as the UCS algorithm, yet their final costs differed due to variations in how costs are calculated with this algorithm.

The UCS algorithm worked really effectively. It helped me find the best solution, which turned out to be the most budget-friendly option.