**Unit 4 Algorithmics**

**Week 7 Submit Questions**

1. Here, we consider the minimum map colouring problem where we seek to find how to colour a map using as few colours as possible so that no two adjacent nodes share the same colour.

1. How is this different to the 3-colour map problem?

Minimum map colouring problem seeks to find the smallest number of colours needed to colour a map such that no two adjacent nodes (regions) share the same colour. In contrast, the 3-colour map problem is a specific instance of the map colouring problem where we determine if a map can be coloured using exactly three colours, again ensuring that no two adjacent nodes share the same colour.

1. Which problem class does the minimum map colouring problem belong to?

NP-Hard, this means that there is no known polynomial-time algorithm to solve this problem

1. Describe a deterministic approach to solving the minimum map colouring problem. What is the time complexity of your approach?

 Assign the first colour to the first node.

 Move to the next node and assign the first available colour that has not been used by its adjacent nodes.

 If no available colour can be assigned, backtrack to the previous node and try the next colour.

 Repeat steps 2-3 until all nodes are coloured or no solution is found (which means increasing the number of colours).

The time complexity of the backtracking approach is O(m^n), where m is the number of colours and n is the number of nodes.

1. Explain how we could use simulated annealing to solve this problem. How could this lead to an advantage over the deterministic method you describe above?

 Start with an initial solution (a colouring of the map).

 At each iteration, randomly select a node and change its colour to a different one.

 Calculate the change in the "energy" (number of adjacent nodes with the same colour).

 If the new solution is better (lower energy), accept it.

 If the new solution is worse, accept it with a probability that decreases over time (controlled by a "temperature" parameter that gradually decreases).

 Repeat steps 2-5 until the system "cools down" to a low temperature.

2. Consider the A\* algorithm.

1. Explain the algorithm it improves and how it does so.

While Dijkstra's algorithm finds the shortest path from a starting node to all other nodes in a graph, A\* focuses on finding the shortest path from a starting node to a specific goal node. It does so by using a heuristic to estimate the cost from the current node to the goal, which helps guide the search and reduces the number of nodes that need to be explored.

1. What property of this (or any other) heuristic must be maintained in order to guarantee the optimality of the heuristic?

It must not overestimate the actual cost to reach goal node from any other node and be consistent, meaning the estimated cost is always less than or equal to the actual cost from the current node to a neighbor plus the estimated cost from the neighbor to the goal, then A\* is guaranteed to find the optimal path.

3. Consider the Travelling Salesman Problem. You are working for a delivery app startup who require you to determine a route that a delivery driver must take in order to quickly deliver their parcels to a given list of different locations. They require you to design an algorithm that finds such a path in reasonable time.

1. Explain why it is not possible for you to guarantee that your solution is the optimal one.

The Travelling Salesman Problem (TSP) is an NP-hard problem, meaning there is no known polynomial-time algorithm to solve it exactly. Because of this, it is computationally infeasible to guarantee an optimal solution for large instances of the problem in reasonable time.

1. Explain how you could use a greedy algorithm to quickly find one solution.

 Start at an arbitrary node (city).

 At each step, move to the nearest unvisited node.

 Repeat until all nodes have been visited.

 Return to the starting node to complete the tour.

1. Explain how you could use hill-climbing to find a better solution.

 Start with an initial solution (a tour).

 Evaluate the cost (length) of the current tour.

 Generate neighboring solutions by making small changes to the current tour (e.g., swapping two cities).

 Move to the neighbor with the lowest cost if it improves the solution.

 Repeat steps 3-4 until no further improvements can be made.

d. Explain whether or not your hill-climbing approach will work in reasonable time.

t is more efficient than exhaustive search methods like backtracking or brute force, which have factorial time complexity. However, its performance significantly depends on the size of the problem and the nature of the initial solution. Through it is not guaranteed to find the optimal solution and can get stuck in local optima.