**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?

The 3-colour map problem is a decision problem: given any map, can it be coloured using only three colours? This returns a boolean value, while the minimum map colouring problem returns an integer for the minimum number of colours needed to colour the map. Here, colouring a map refers to the process of assigning each node an individual colour such that no adjacent nodes have the same colour. Since the 3-colour map problem is a decision problem, it is NP-complete, because a 3-colour solution can be verified in polynomial time, whereas the minimum map colouring problem is NP-Hard.

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

As stated above, this problem is NP-Hard.

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

The problem could be solved with brute force. Namely, the algorithm could start with a variable . Each node would be assigned a colour from a set containing colours, and the search space would be explored using backtracking. If a solution is found, it is returned and the program terminates, returning the value of and the graph colouring, and if a solution is not found, the value is incremented by 1 and the above process is repeated. Since in the worst case, each node needs to be checked as different colours in each iteration, resulting in a Big O of where is the number of nodes. Since we are looking at the asymptotic time complexity, this simplifies to .

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?

We could use simulated annealing by starting with an initial max\_colouring set to the number of nodes there are in the graph. This initial candidate solution could then be randomly permutated by simply changing the colour of a random node to a different colour. A cost function could be implemented that returns the number of edges that connect nodes with the same colour, with a cost of 0 being ideal (being a valid graph). After a set amount of iterations, simulated annealing would lower the max\_colouring value and see if can find a valid graph colouring (cost function = 0) with the lower max\_colouring. If it can, it will continue trying to lower the value, and if it cannot, it will return the previous lowest max\_colouring. Of course, at each instance of max\_colouring, the normal simulated annealing rules would apply, where the initial temperature would be high, meaning that solutions with a higher cost function would be accepted, but the temperature will decrease over time such that the algorithm can hopefully converge on the lowest possible cost for that given value max\_colouring, ideally being 0. This could be better than the deterministic method for extremely large graphs/maps, as it will provide an approximate minimum number of colours but will be much faster. The efficiency could be increased by initially decreasing the max\_colouring value by large increments and then by smaller increments upon unsuccessful attempts.

2. Consider the A\* algorithm.

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

A\* attempts to improve upon Dijkstra’s algorithm. It does this by providing the algorithm with a sense of “how close it is to the goal”. This is done by a heuristic function, , which estimates the cost of moving from the current node to the target node. Dijkstra’s only uses , the cost of moving from the source node to the current node, but A\* adds this value up with as well to create a combined function . Now, instead of selecting the next node to visit based on the lowest value of like Dijkstra’s, A\* uses the lowest value of . This is an improvement because it will prioritise the search space that is closer to the goal, meaning it will be found faster.

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

The heuristic must not overestimate the distance to the goal node, if not it is not gaurenteed to produce an 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.

It is very unlikely that it is possible for me to guarantee an optimal solution in a reasonable time because the Travelling Salesman Problem is NP-Hard. This means that the TSP, as far as we know so far is intractable, but since P is a subset of NP-Hard, it could very well be possible that there *is* a polynomial time solution to the TSP, but that is quite unlikely.

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

The Nearest Neighbour Heuristic could be used to find a possible candidate solution, where the closest unvisited node is selected from any given node, which will create a tour of the graph quite fast. It is worth noting that this solution will most likely be suboptimal, as a greedy approach is not well suited to the TSP.

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

Hill-Climbing could be used by utilising 2-opt switches randomly. Using this technique, we would start with the initial candidate solution generated by the NN heuristic and then apply random 2-opt switches to it and see if that has improved the cost or not. If the cost has been improved, we take the new candidate, and if it has not then we continue with the current candidate.

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

The Hill-Climbing approach will work in reasonable time, because the NN Heuristic is a polytime solution, albeit a very good one, and the Hill-Climbing optimisation can be done in linear time, as the main time cost is simply calculating the cost of the tour. Therefore, because a combination of the NN Heuristic and HC can be done in linear time, it will be much more reasonable than an exact algorithm.