Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 9 - Approximation Algorithms

1. Recall the VERTEX COVER problem from the lectures, which we proved to be NP-complete. We will consider several greedy algorithms for the problem.

   GREEDY1: Repeat the following process until the graph is empty: consider any edge (v_1, v_2) of the graph G and include either v_1 or v_2 in the vertex cover S. Remove the chosen node and all its incident edges from the graph.

   GREEDY2: Repeat the following process until the graph is empty: consider any edge (v_1, v_2) of the graph G and include both of v_1 and v_2 in the vertex cover S. Remove v_1 and v_2 from the graph and all their incident edges.

   GREEDY3: Start with an empty set S. While S is not a vertex cover, choose a node u which maximises the number of uncovered edges incident to it and add it to S.

   (a) Show that the approximation ratio of GREEDY1 is $\omega(1)$.

   (b) Show that the approximation ratio of GREEDY2 is at most 2.

   (c) Does GREEDY3 also achieve an approximation ratio of 2? Justify your answer.

2. A family is moving house, and they would like to move all of their belongings. To do that, they would like to purchase as few boxes as possible, to transfer all of their belongings. In particular, they have n items, with item i having size $w_i \in [0, 1)$, and each box has size 1. They would like to put all of the items in some box, and use the minimum number of boxes possible.

   (a) The family consider the following greedy algorithm for their problem: They fix an ordering of their items and open a box. They start putting items in the box until the next item in the order does not fit. When this happens, they open a new box and continue with the same process. The process terminates when all the items have been put in boxes.

   Prove that this greedy algorithm has approximation ratio 2, i.e., it uses twice as many boxes as the optimal solution. Argue about the correctness of your algorithm, by providing an analysis for the upper bound, and a tight example for the lower bound.

   (b) Prove that the problem faced by the family is NP-hard, by providing a polynomial-time reduction from the following problem:

   PARTITION: Given positive integers $\alpha_1, \alpha_2, \ldots, \alpha_n$, decide whether it is possible to find a subset A of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in A} \alpha_i = \sum_{i \notin A} \alpha_i$.

3. *(Optional, more advanced)* Consider the SET COVER problem:

Definition 1 (Weighted Set Cover)**.** We are given a set of elements $\mathcal{U}$ and a collection of subsets of $\mathcal{U}$, each with an associated weight w_i, whose union is $\mathcal{U}$. The goal is to find set of subsets of $\mathcal{U}$ whose union is equal to $\mathcal{U}$, of minimum total weight, i.e., a set of sets $S = S_1, \ldots, S_k$ such that for every element $j \in U$, there exists a set $S_i \in S$ such that $j \in S_i$., such that $\sum_{i \in S} w_i$ is minimised.

Show that Set Cover is a generalization of Vertex Cover.

Consider the following greedy algorithm for Set Cover: Greedily select sets based on the following "efficiency measure": Let's assume that we have already covered some elements of $\mathcal{U}$ by our selection of sets, and let R be the remaining elements of $\mathcal{U}$ to be covered. Among the sets that have not been yet selected, we will select that which minimises

$$\frac{w_i}{|S_i \cap R|}.$$

The idea is that we want to select sets that have small weight, but also cover lots of elements[1]. The pseudocode for the algorithm is the following.

Algorithm 1 Greedy Weighted Set Cover

```
1: Start with $R = \mathcal{U}$ and $\mathcal{C} = \emptyset$.          ▷ Note that $\mathcal{C}$ is a set of sets.
2: while $R \neq \emptyset$ do
3:     Select a set $S_i \in \arg\min \frac{w_i}{|S_i \cap R|}$.
4:     Add $S_i$ to $\mathcal{C}$.
5:     Remove the elements of $S_i$ from $R$          ▷ These elements have now been covered.
6: Return $\mathcal{C}$.
```

Prove that the approximation ratio of this greedy algorithm is $O(\lg n)$, where $n = |\mathcal{U}|$.

[1]This might be reminiscent of the greedy algorithm we used for 0/1-Knapsack, where we wanted elements that have large value but small weight, and we took the ratio v_i/w_i as our measure of efficiency.