INTRO TO APPROXIMATION, CLASS 1

learning approximations, vertex cover

D(approximation algorithm**):** Suppose we have a polynomial-time algorithm for a minimization problem. Then, we call it an *c-approximation algorithm*, if for every instance of the problem I we can bound the value of the output of the algorithm $ALG(I)$ by c times the value of the optimal solution of the problem $OPT(I)$; in other words $ALG(I) \leq c \cdot OPT(I)$.

For maximization problems there is a slightly different equation and a different range of the parameter c: there we have $\forall I\colon ALG(I) \geq c \cdot OPT(I)$.

Therefore, we sometimes design and analyze $\frac{1}{2}$-approximation algorithms and sometimes 2-approximation algorithms.

Our goal (most of the time) is for c to be a small constant, but it will not be possible all the time. It is therefore valid to design $\frac{1}{n}$-approximation or $O(\log n)$-approximation algorithms.

Some algorithms are simple and have a constant approximation ratio:

Exercise one   Consider the problem Maximum Planar Subgraph. On input we get a connected non-planar graph G and our task is to find a planar subgraph $R \subseteq G$ – while maximizing the number of remaining edges in R.

Design a 1/3-approximation algorithm for this problem.

However, some simple algorithms do not perform well:

Exercise two   Consider the following approximation algorithm for Maximum Clique:

„For every edge $e \in E(G)$ we greedily test whether we can find a third vertex which forms a triangle with the endpoints of e. If we find it, we search for a fourth vertex which forms a K_4 with the three previously selected vertices.

We continue until we cannot add any more vertices to this clique. Then we stop and do the same procedure for another edge $e' \in E(G)$. We output the biggest clique that the algorithm found.“

Your task: to establish some bounds for the approximation ratio of this algorithm.

For some problems we can even show that an approximation algorithm is very unlikely:

Exercise three   Consider the classic optimization problem Bin Packing. Here we get on input n items, each with size in $[0, 1]$. Our goal is to pack these items into bins, where each bin has capacity 1. There is no limit on the number of bins, but we want to minimize the number of non-empty bins that we use.

Prove that if there was a 1.499-approximation algorithm for this problem, then $P = NP$.

Hints:

- Since we want to prove hardness, a possible argument could look like this:
  „We take every input instance of some NP-complete problem, we transform them into instances for Bin Packing and if the assumed 1.499-approximation algorithm packs the instance, we can tell from the output whether this was a *Yes*-instance or a *No*-instance of the original NP-complete problem.“
- We can reduce from any NP-complete problem we want. It is often useful to first try either the classical ones (3-SAT), or ones that have similarities to our current problem – in this case they may work with numbers, or involve partitioning objects into groups.
  Make a list of 3-5 problems which you would consider as a good start for a reduction.

- Consider the following: if the optimum uses only k bins, then a 1.499-approximation algorithm can use only at most $\lfloor 1.499k \rfloor$ bins.

In the rest of the exercises we focus on the problem MINIMUM VERTEX COVER. A *vertex cover* is a subset of vertices of a graph such that for every edge of the graph, at least one endpoint of the edge lies in the vertex cover. We search for the vertex cover with as few vertices as possible. If the vertices are given non-negative weights, we would look for the MINIMUM WEIGHTED VERTEX COVER.

EXERCISE FOUR

Consider the following greedy algorithm:

„Choose any edge in the graph. We need to cover it using at least one of its endpoints – but we select both of them and add them to the cover.

We remove the two vertices (and all their adjacent edges) and continue by selecting another edge.“

Does this algorithm have a constant approximation ratio?

EXERCISE FIVE

- Try to recall linear programming classes and formulate MINIMUM VERTEX COVER as an integer program. Start with binary variables x_v for every vertex $v \in V(G)$ …
- The basic linear programming classes usually start with an integer program and quickly move on to the linear programming relaxation, which is solvable in polynomial time. However, to make an approximation algorithm we need to convert optimal linear solutions to good (but non-optimal) integer solutions.
  Suggest a simple approach how to convert the optimum of the linear program for MINIMUM VERTEX COVER to a *feasible* integer solution, which will also be a 2-approximation.

EXERCISE SIX

Finally, we consider MINIMUM WEIGHTED VERTEX COVER – as mentioned before, we have non-negative weights $w_v \geq 0$ for every vertex v and we search for the vertex cover with minimum total weight, not necessarily minimum number of vertices.

1. How good is the greedy two-vertex-selecting algorithm that we analyzed above?
2. Design a 2-approximation algorithm for MINIMUM WEIGHTED VERTEX COVER.