University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
S. Faour, M. Fuchs, G. Schmid

Algorithm Theory
Exercise Sheet 3

Due: Wednesday, 16th of November, 2022, 11:59 pm

Exercise 1: Quick Greedy Analysis *(7 Points)*

Given the following problems and proposed greedy strategies either formally prove or disprove (e.g. by counterexample) that the proposed greedy strategy is optimal for the given problem:

1. **Problem: Minimum Dominating Set** *(2 Points)*
   In an undirected graph $G = (V, E)$, with $n = |V|$.
   We want to find the smallest set S such that every node $v \in V$ either is in S, or has a neighbor in S (we say v is dominated by S).
   Or equivalently:

$$\forall v \ \in V : \ v \in S \vee \exists u \in S : (v, u) \in E$$

   Strategy: Pick node that covers the most undominated neighbors
   Nodes can be coloured in 3 colours: *white*, *black* or *grey*. *White* represents nodes that are not yet dominated. *Black* represents nodes that are already part of our dominating set and *grey* represents nodes that are adjacent to a *black* node.
   At the beginning all nodes are *white*. Then until no more *white* nodes exist, pick the node v^* that has the most *white* neighbous, or formally:

$$v^* = \arg\max_{v \in V} \deg_W(v)$$

   where $\deg_W(v)$ is the number of *white* nodes adjacent to v.
   Then color v^* *black* and all of the nodes adjacent to v^* *grey*. Repeat until no *white* nodes remain.
   At the end all of the *black* nodes together form the Dominating Set.

2. **Problem: Most finished jobs** *(3 Points)*
   We want to schedule n jobs $a_1, a_2, \ldots, a_n$ with durations $d_1, d_2, \ldots, d_n$. A job a_i is considered finished at time t if its starting time s_i fulfils the condition $s_i + d_i \leq t$.
   At each point in time there can be only one job currently worked on. We define this condition formally as: compute $s_1, s_2, ..., s_n$ such that they satisfy

$$\forall i, j \in \{1, \ldots, n\}, j \neq i : s_j \notin \{s_i, s_i + 1, \ldots, s_i + d_i\}$$

   or in words, that no job starts while another job is currently being worked on.
   Our goal is to compute $s_1, s_2, ..., s_n$ such that at every point in time t the number of already finished jobs is maximized.

   Strategy: Pick shortest job
   Whenever possible, meaning that whenever there is no job currently being worked on, pick the job with the shortest duration, that is not yet finished.

3. **Problem: Max-Cut** *(2 Points)*
   In an undirected graph $G = (V, E)$, with $n = |V|$.
   We define a cut $S \subset V$ to be a separation of V into two distinct subsets S and $V \backslash S$. We define the weight of a cut $w(S) = |\{(u, v) \mid u \in S, v \in V \backslash S\}|$ to be the number of edges between the two sets S and $V \backslash S$.
   The goal is to find a maximum weight cut.

   Strategy: Pick node that increases the cut the most
   We define the relative-degree of $v \in V$ with respect to a subset $A \subset V$ to be

$$d_A(v) = |\{u \in A \mid (v, u) \in E\}|$$

   being the number of edges from v to nodes in A.

   Again we start with an empty set $S = \emptyset$ and in every step try to get the biggest increase to our cut-weight. For this we either add a node v^* to our set $S = S \cup \{v^*\}$, or if already $v^* \in S$ we remove it from $S = S \backslash \{v^*\}$, where we determine v^* by the following characteristic:

$$v^* = \arg\max_{v \in V} \left\{ \begin{array}{ll} d_S(v) - d_{V \backslash S}(v), & \text{for } v \in S \\ d_{V \backslash S}(v) - d_S(v), & \text{for } v \in V \backslash S \end{array} \right\}$$

   In words, we pick a v^* such that the increase to the cut would be biggest. Hence every such step will give us an improvement to our cut weight.
   We stop once no $v \in V \backslash S$ can give us an improvement.

Exercise 2: Greedy Satisfying *(13 Points)*

We look at a boolean formula ϕ of n clauses in 3-Conjunctive Normal Form (CNF), meaning that it is given as a "logical and":$\wedge$ of clauses that only use "logical or's":$\vee$. In each clause we have exactly 3 literals, so an example would be:

$$\phi = (a \vee \bar{b} \vee c) \wedge (\bar{c} \vee d \vee e) \wedge (\bar{a} \vee \bar{d} \vee \bar{b})$$

Note: $\bar{a}$ stands for the negation of the variable a.

Devise an algorithm based on a greedy strategy to satisfy as many clauses as possible. Give an explanation why your algorithm is a greedy algorithm.
Make an effort to make the algorithm efficient! Argue the runtime of your algorithm, the runtime of your algorithm must be at most polynomial in n. You will most likely not be able to devise an algorithm that can satisfy all clauses/ that is optimal, since this problem is NP-hard. Therefor give an algorithm that is "as good as possible".

Prove a performance guarantee for your algorithm. More specifically this means, that you should prove that your algorithm can not be arbitrarily bad. Give a bound on $\frac{1}{2} < \alpha \leq 1$, such that your algorithm satisfies at least an α-fraction of clauses.

Hint: For the performance analysis, it might be helpful to think about the number of literals that have not been assigned thus far and look at how this number changes after assigning a literal.