General rules for solving exercises

- This is a theory course, which means: if an exercise does not explicitly say “you do not need to prove your answer” or “justify intuitively”, then a formal proof is always required.

- All exercises and their solutions, no matter whether they are graded or regular/optional ones, are part of the material relevant for the two exams.

- Some of the exercises are marked as "in-class", which means that we do not expect you to solve them before the exercise session. Instead, your teaching assistant will solve them with you in class.

- You are highly encouraged to solve all other exercises (those not marked as "in-class") on your own and to hand in a writeup of your solutions no later than the due date. If you choose to do so, please write the name of your teaching assistant on the front sheet.

Solution 1: Solving Linear Programs via Binary Search

Suppose we are given a linear program $L$ that asks us to maximize the objective function $c^T x$ where $c \neq 0$. Minimization problems can be dealt with analogously, and problems with $c = 0$ can be solved with one single call to an algorithm that solves the feasibility problem (because any feasible solution is also an optimal solution).

As we will see, with binary search we will not manage to find an optimal solution, but we can get arbitrarily close. So, suppose $x^*$ is an optimal solution to the given linear program $L$ with objective value $\text{OPT} := c^T x^*$. Our goal is to find an approximate solution $\tilde{x}$ that satisfies $\text{OPT} - c^T \tilde{x} \leq \epsilon$, where $\epsilon > 0$ is an arbitrary but fixed error term.

In order to perform binary search, we need to initialize and maintain upper and lower bounds on the optimum objective value $\text{OPT}$. For this we will need a stronger version of Theorem 6.2, which says that there exists an optimal solution, let us call it $x$, without loss of generality, that is contained in the cube $[-K, K]^n$ with $K \leq 2^{O(|L|)}$. We note that the proof in the lecture notes already implies this stronger statement.

It is an easy task to find the vertex $x_{\text{max}}$ (resp., $x_{\text{min}}$) of the cube $[-K, +K]^n$ which maximizes $c^T x_{\text{max}}$ (resp., minimizes $c^T x_{\text{min}}$). Indeed, the sign of any coordinate of $c$ corresponds to the sign of the corresponding coordinate of $x_{\text{max}}$. Also, clearly, $x_{\text{min}} = -x_{\text{max}}$. Since, as we said earlier, $x^*$ is contained in $[-K, K]^n$ we get that $\alpha := c^T x_{\text{max}}$ and $\beta := c^T x_{\text{min}}$ are upper and lower bounds, respectively, for $\text{OPT}$.

Now we can perform binary search for $\text{OPT}$. That is, we let $\gamma := \frac{1}{2}(\alpha + \beta)$ and we add the constraint $c^T x \geq \gamma$ to $L$. We check whether the new program is still feasible. If it is, then we update the lower bound $\beta := \gamma$. If it is not, then we remove the new constraint again and we update the upper bound $\alpha := \gamma$. In any case, the size of the interval $[\beta, \alpha]$ that contains $\text{OPT}$...
halves in every step of the search. Therefore, the number of steps until we reach $\alpha - \beta \leq \epsilon$
(and therefore also our goal $\text{OPT} - \beta \leq \epsilon$) is at most

$$\log_2 \frac{cT_{\max} - cT_{\min}}{\epsilon} = \log_2(2^{O(\langle L \rangle)}) - \log_2(\epsilon) = O(\langle L \rangle),$$

for any fixed $\epsilon > 0$.

**Solution 2: Rotation Matrices**

We recall from linear algebra that a matrix $R \in \mathbb{R}^{n \times n}$ is a rotation matrix if and only if $R^T = R^{-1}$ (in other words, the columns of $R$ form an orthonormal basis of $\mathbb{R}^n$) and if $\det R = 1$ (if $\det R = -1$ then what we have instead is a rotation combined with a reflection). So, let $v_1, \ldots, v_n \in \mathbb{R}^n$ be any set of normalized and pairwise orthogonal vectors with $v_1 = v$. Then, $R := (v_1, \ldots, v_n) \in \mathbb{R}^{n \times n}$ is a rotation matrix which obviously satisfies $R_1 = v_1 = v$, provided that $\det R = 1$. If $\det R = -1$ then we simply replace the vector $v_n$, say, with $-v_n$.

**Solution 3: Deciding Feasibility vs. Finding Feasible Solutions**

First, we check with one call to the oracle whether the given system of linear inequalities has a solution. If the answer is NO then we stop and also output NO. If the answer is YES then we proceed as follows.

(a) If there are only equations in the system, then this just means that we have a system $Ax = b$ of linear equations, for which we can find a solution in polynomial time by Gauss elimination.\(^1\) So, in this case we need no additional calls to the oracle.

(b) If there is at least one inequality, say $ax \leq b$, then we replace it by $ax = b$. If the new, more constrained, system still has a solution (which can be checked with one additional call to the oracle), then we can recursively find a solution to the original system by finding a solution to the more constrained system. If the new, more constrained, system turns out to have no solution, then we drop the constraint $ax \leq b$ completely to obtain a smaller system, which again can be solved recursively. This is a sound strategy because replacing a constraint $ax \leq b$ with $ax = b$ can turn a feasible problem into an infeasible one if and only if the hyperplane corresponding to $ax \leq b$ is not part of the boundary of the feasible region (in other words, it is redundant).

Since we need one initial call to the oracle, and exactly one call per inequality that we get rid of (either by replacing it with an equality or by dropping it completely), the total number of calls to the oracle will be $m + 1$, where $m$ is the number of inequalities in the original system.

**Solution 4: Vertex Cover**

The LP-relaxation of minimum vertex cover with unit costs looks as follows.

\[\begin{align*}
\text{(P):} & \quad \text{Minimize } \sum_{v \in V} x_v \\
& \quad \text{subject to } x_v \geq 0 \quad \forall v \in V \\
& \quad x_u + x_v \geq 1 \quad \forall \{u, v\} \in E
\end{align*}\]

\(^1\)When the computation has to be done exactly, naive implementations of Gauss elimination can lead to an exponential blow-up of the encoding size of intermediate results. However, there are more clever implementations which do not have this problem and which do run in polynomial time.
From the lecture we know that any basic feasible optimal solution $x^*$ for (P) satisfies $x^*_v \in \{0, 0.5, 1\}$ for all $v \in V$. We propose the following rounding algorithm that converts such an $x^*$ into an integral solution $\tilde{x}$: If $x^*_v = 0$ then we put $\tilde{x}_v := 0$. If $x^*_v = 1$ then we put $\tilde{x}_v := 1$. If there exists at least one $v$ with $x^*_v = 0.5$ then we pick exactly one such vertex, let us call it $\hat{v}$, arbitrarily and we put $\tilde{x}_\hat{v} := 0$. For all other $v \neq \hat{v}$ with $x^*_v = 0.5$ we put $\tilde{x}_v := 1$.

It is not hard to see that $\tilde{x}$ again encodes a vertex cover. The only difference to the lecture are edges that contain the special vertex $\hat{v}$ (assuming it exists at all). For any such edge $\{\hat{v}, v\}$ we must have either $x^*_v = 0.5$ or $x^*_v = 1$ since otherwise not all constraints of (P) would be satisfied. Since by construction $\hat{v}$ is the only vertex for which we have $\tilde{x}_\hat{v} < x^*_\hat{v}$, we can conclude that $\tilde{x}_v = 1$ and that the edge $\{\hat{v}, v\}$ is properly covered by the vertex cover encoded by $\tilde{x}$.

As for the approximation ratio, we see that if there is no vertex $v$ with $x^*_v = 0.5$ (in other words, if the special vertex $\hat{v}$ does not exist), then $\tilde{x} = x^*$ and thus $\tilde{x}$ must be an optimal integral solution. Otherwise, similar to the lecture, we derive

$$\text{cost}(\tilde{x}) := \sum_{v \in V} \tilde{x}_v \leq 2 \sum_{v \in V} x^*_v - 1 \leq 2 \cdot \text{OPT} - 1 \leq \left(2 - \frac{1}{\text{OPT}}\right) \cdot \text{OPT} \leq \left(2 - \frac{1}{n}\right) \cdot \text{OPT},$$

where $\text{OPT}$ is the cost of an optimal vertex cover and where the last inequality uses the obvious fact $\text{OPT} \leq n$. 

3