ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Midterm Exam, Advanced Algorithms 2016-2017

- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudo-code without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- You are allowed to refer to algorithms covered in class without reproving their properties.
- **Do not touch until the start of the exam.**

Good luck!

Name: ____________________ **N° Sciper:** ____________

Problem 1 / 30 points	Problem 2 / 20 points	Problem 3 / 20 points	Problem 4 / 30 points

Total / 100

1 *(consisting of subproblems **a**-**b**, 30 pts)* **Basic questions.** This problem consists of two subproblems that are each worth 15 points.

1a *(15 pts)* Suppose we use the Simplex method to solve the following linear program:

$$\begin{aligned}
\textbf{maximize} \quad & 2x_1 - x_2 \\
\textbf{subject to} \quad & x_1 - x_2 + s_1 = 1 \\
& x_1 + s_2 = 4 \\
& x_2 + s_3 = 2 \\
& x_1,\, x_2,\, s_1,\, s_2,\, s_3 \geq 0
\end{aligned}$$

At the current step, we have the following Simplex tableau:

$$\begin{aligned}
x_1 &= 1 + x_2 - s_1 \\
s_2 &= 3 - x_2 + s_1 \\
s_3 &= 2 - x_2 \\
\hline
z &= 2 + x_2 - 2s_1
\end{aligned}$$

Write the tableau obtained by executing one iteration (pivot) of the Simplex method starting from the above tableau.

Solution: At the current step, we have the following Simplex tableau:

$$\begin{aligned}
x_1 &= 1 + x_2 - s_1 && (1) \\
s_2 &= 3 - x_2 + s_1 && (2) \\
s_3 &= 2 - x_2 && (3) \\
\hline
z &= 2 + x_2 - 2s_1 \\
x_1 &:= 1 \; x_2 := 0 \; s_1 := 0 \; s_2 := 3 \; s_3 := 2
\end{aligned}$$

Only x_2 has a positive coefficient in z, we will pivot x_2. We have
$\nearrow x_2 \longrightarrow \; x_2 \leq \; \infty \; (1), \; x_2 \leq 3 \; (2), \; x_2 \leq 2 \; (3) \longrightarrow x_2 := 2, \; s_3 := 0$

$$\begin{aligned}
x_1 &= 3 - s_1 - s_3 \\
s_2 &= 1 + s_1 + s_3 \\
x_2 &= 2 - s_3 \\
\hline
z &= 4 - 2s_1 - s_3 \\
x_1 &:= 3 \; x_2 := 2 \; s_1 := 0 \; s_2 := 1 \; s_3 := 0
\end{aligned}$$

1b *(15 pts)* In the maximum directed cut problem we are given as input a directed graph $G = (V, A)$. Each arc $(i, j) \in A$ has a nonnegative weight $w_{ij} \geq 0$. The goal is to partition V into two sets U and $W = V \setminus U$ so as to maximize the total weight of the arcs going from U to W (that is, arcs (i, j) with $i \in U$ and $j \in W$). Give a randomized 1/4-approximation algorithm for this problem (together with a proof that it is a 1/4-approximation in expectation).

Solution: Assign each vertex with probability 0.5 to either side. Then, for any directed edge (i, j)

$$Pr((i,j) \text{ in cut}) = Pr(i \in U \wedge j \in W) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} \tag{1}$$

The expected total weight of the cut is

$$\sum_{(i,j)\in A} Pr((i,j) \text{ in cut}) = \frac{1}{4} \sum_{(i,j)\in A} w_{ij} \geq \frac{1}{4} OPT \tag{2}$$

2 *(20 pts)* **Spanning trees with colors.** Consider the following problem where we are given an edge-colored graph and we wish to find a spanning tree that contains a specified number of edges of each color:

Input: A connected undirected graph $G = (V, E)$ where the edges E are partitioned into k color classes $E_1, E_2, \ldots, E_k$. In addition each color class i has a target number $t_i \in \mathbb{N}$.

Output: If possible, a spanning tree $T \subseteq E$ of the graph satisfying the color requirements:

$$|T \cap E_i| = t_i \qquad \text{for } i = 1, \ldots, k.$$

Otherwise, i.e., if no such spanning tree T exists, output that no solution exists.

Design a polynomial time algorithm for the above problem. You should analyze the correctness of your algorithm, i.e., why it finds a solution if possible. To do so, you are allowed to use algorithms and results seen in class without reexplaining them.

Solution: We solve this problem using matroid intersection. First observe that if the summation of the t_i for $1 \le i \le k$ is not equal to $n-1$ then there is no feasible solution since we know that the number of edge in any spanning tree is exactly $n-1$. Therefore, we assume $\sum_{1\le i\le k} t_i = n-1$. The ground set for both matroids that we use is the set of the edges E. First matroid that we use is the graphic matroid. The second matroid that we use is a partition matroid with following independent sets:

$$\mathcal{I} = \{F \subseteq E \mid |F \cap E_i| \le t_i, \quad \text{for } 1 \le i \le k\}$$

As shown in class the both above defined matroids are indeed matroid. Now assume that F is the maximum size independent set the intersection of these two matroids (we saw in the class how we can find F). If $|F| < n-1$ it is not possible to find a solution for our problem, since any solution to our problem corresponds to a solution in the intersection of these two matroids of size $n-1$. Moreover, if $|F| = n-1$, than F is a spanning tree and $|F \cap E_i| \le t_i$. Also, we know that $|F| = n-1$ and $\sum_{1\le i\le k} t_i = n-1$ and E_i's are disjoint. Therefore $|F \cap E_i| = t_i$, so we get the desired solution.

(This page is intentionally left blank.)

(Primal) LP Relaxation	(Dual)
minimize $\sum_{S \in \mathcal{T}} c(S) x_S$	**maximize** $\sum_{e \in \mathcal{U}} y_e$
subject to $\sum_{S \in \mathcal{T} : e \in S} x_S \geq 1$ for $e \in \mathcal{U}$	**subject to** $\sum_{e \in S} y_e \leq c(S)$ for $S \in \mathcal{T}$
$x_S \geq 0$ for $S \in \mathcal{T}$	$y_e \geq 0$ for $e \in \mathcal{U}$

Figure 1. The standard LP relaxation of set cover and its dual.

3 *(20 pts)* **Set Cover.** (For this problem, it is good to recall the standard linear programming relaxation of set cover and its dual, see Figure 1.)

You just graduated from EPFL and you started your first inspiring job. To your surprise, you immediately find yourself in the midst of the action as the company's future depends on obtaining a good solution to a set cover instance. The instance is specified by a universe $\mathcal{U} = \{e_1, \ldots, e_n\}$ of clients, a family of subsets $\mathcal{T} = \{S_1, S_2, \ldots, S_m\}$, and a cost function $c : \mathcal{T} \rightarrow \mathbb{R}_+$. The task is to find a collection $C \subseteq \mathcal{T}$ of subsets of minimum total cost that covers all elements.

After spending a good month on the problem, you have found a super heuristic that outputs (what you think) is a great solution C^*. However, as it turns out, your boss studied mathematics back in the 60's and so she is not convinced by your heuristic arguments. You therefore contact EPFL, who use their supercomputers to compute a dual certificate. They found a dual certificate y^*, which is a feasible solution to the dual of the LP relaxation of set cover satisfying the following:

- For every $S \in C^*$, we have $\sum_{e \in S} y^*_e \geq c(S)/2$.
- For every $e \in \mathcal{U}$ with $y^*_e > 0$, we have $|\{S \in C^* : e \in S\}| \leq 2$.

Using the dual solution y^*, **prove that your solution C^* is in fact a 4-approximate solution**. Since the set cover problem is NP-hard to approximate within a factor better than $\ln(n)$ in general, the fact that you have found a much better solution for the given instance will rescue the company and almost guarantee you a promotion.

Solution: Let us compute the cost of solution C^* denoted by $cost(C^*)$.

$$cost(C^*) = \sum_{S \in C^*} c(S).$$

By the first property of dual certificate we get that

$$cost(C^*) = \sum_{S \in C^*} c(S) \leq 2 \sum_{S \in C^*} \sum_{e \in S} y^*_e.$$

By the second property of the dual certificate we get that for any element $e \in \mathcal{U}$, either y^*_e appears exactly twice in the above summation or $y^*_e = 0$ so it does not matter how many times it appears on the summation. Therefore, we get

$$cost(C^*) = 2 \sum_{S \in C^*} \sum_{e \in S} y^*_e \leq 2 \cdot 2 \sum_{e \in \mathcal{U}} y^*_e.$$

Since we know that y^* is a feasible solution of the dual then $\sum_{e \in \mathcal{U}} y_e^* \leq$ OPT-LP, where OPT-LP is the optimum solution to the above linear program. Moreover since the above linear program is a relaxation for Set Cover problem, we get that:

$$cost(C^*) \leq 4 \cdot \text{OPT}$$

where OPT is the optimum set cover solution. This shows that C^* is a 4-approximation solution.

(This page is intentionally left blank.)

4 *(consisting of subproblems **a**-**b**; note that you can solve subproblem **b** assuming the solution to **a** without solving it; 30 pts)*

Efficient job scheduling. In this problem we are going to investigate the linear programming relaxation of a classical scheduling problem. In the considered problem, we are given a set M of m machines and a set J of n jobs. Each job $j \in J$ has a processing time $p_j > 0$ and can be processed on a subset $N(j) \subseteq M$ of the machines. The goal is to assign each job j to a machine in $N(j)$ so as to complete all the jobs by a given deadline T. (Each machine can only process one job at a time.)

If we, for $j \in J$ and $i \in N(j)$, let x_{ij} denote the indicator variable indicating that j was assigned to i, then we can formulate the scheduling problem as the following integer linear program:

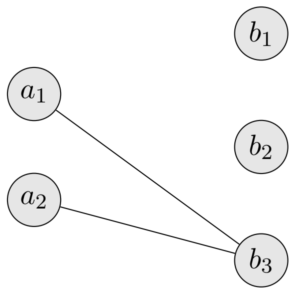
$$\sum_{i \in N(j)} x_{ij} = 1 \qquad \text{for all } j \in J \qquad \textit{(Each job j should be assigned to a machine $i \in N(j)$)}$$

$$\sum_{j \in J : i \in N(j)} x_{ij} p_j \leq T \qquad \text{for all } i \in M \qquad \textit{(Time needed to process jobs assigned to i should be $\leq T$)}$$

$$x_{ij} \in \{0,1\} \text{ for all } j \in J,\ i \in N(j)$$

The above integer linear program is NP-hard to solve, but we can obtain a linear programming relaxation by relaxing the constraints $x_{ij} \in \{0,1\}$ to $x_{ij} \in [0,1]$. The obtained linear program can be solved in polynomial time using e.g. the ellipsoid method.

Example. An example is as follows. We have two machines $M = \{m_1, m_2\}$ and three jobs $J = \{j_1, j_2, j_3\}$. Job j_1 has processing time $1/2$ and can only be assigned to m_1; job j_2 has processing time $1/2$ and can only be assigned to m_2; and job j_3 has processing time 1 and can be assigned to either machine. Finally, we have the "deadline" $T = 1$. An extreme point solution to the linear programming relaxation is $x^*_{11} = 1, x^*_{22} = 1, x^*_{13} = 1/2$ and $x^*_{23} = 1/2$. The associated graph H (defined in subproblem **a**) can be illustrated as follows:



4a *(15 pts)* Let x^* be an extreme point solution to the linear program and consider the (undirected) bipartite graph H associated to x^* defined as follows. Its left-hand-side has a vertex a_i for each machine $i \in M$ and its right-hand-side has a vertex b_j for each job $j \in J$. Finally, H has an edge $\{a_i, b_j\}$ iff $0 < x^*_{ij} < 1$.

Prove that H is acyclic (using that x^* is an extreme point).

Solution:

Suppose that H contains a (simple) cycle. As usual, we will show that x^* can then be written as a convex combination of two different feasible points, contradicting that x^* is an extreme point. Namely, we will define a nonzero vector y such that $x - \epsilon y$ and $x + \epsilon y$ are both feasible (for a small enough $\epsilon > 0$). Of course, $x = \frac{(x-\epsilon y)+(x+\epsilon y)}{2}$.

As in the proof for the bipartite matching polytope, label edges in the cycle alternately as odd and even. In that proof, we defined y to be $+1$ on even edges and -1 on odd edges, but this doesn't quite work here. This is because of the constraints $\sum_{j \in J: i \in N(j)} x_{ij} p_j \leq T$: the left-hand side would change when adding y because the p_j's are different.[1] To deal with this, we instead set $y_{ij} = \pm 1/p_j$ for edges $\{i, j\}$ in the cycle. Then:

- for each $j \in J$ we have $\sum_{i \in N(j)} y_{ij} = 0$ because this sum has two nonzero terms, equal to $1/p_j$ and $-1/p_j$ (or no nonzero terms if j does not appear in the cycle),
- for each $i \in M$ we have $\sum_{j \in J: i \in N(j)} y_{ij} p_j = 0$ because this sum has two nonzero terms, equal to $1/p_j \cdot p_j$ and $-1/p_{j'} \cdot p_{j'}$ where j and j' are the neighbors of i on the cycle (or no nonzero terms if i does not appear in the cycle),
- since $y_{ij} \neq 0$ only if $0 < x^*_{ij} < 1$, we can set ϵ so that $0 \leq x^*_{ij} \pm \epsilon y_{ij} \leq 1$ for all i and j.

This shows that the points $x \pm \epsilon y$ are both feasible.

[1] It is not enough to set ϵ small enough. Consider an instance with two machines and two jobs. The jobs have processing times 1 and 2, and $T = 1.5$. Let x assign $1/2$ of each job to each machine. Then just adding/subtracting $\pm\epsilon$ on the cycle-edges will overload one of the machines, no matter what $\epsilon > 0$ is.

4b *(15 pts)* Use the structural result proved in the first subproblem to devise an efficient rounding algorithm that, given an instance and a feasible extreme point x^* in the linear programming relaxation corresponding to the instance, returns a schedule that completes all jobs by deadline $T + \max_{j\in J} p_j$. In other words, you wish to assign jobs to machines so that the total processing time of the jobs a machine receives is at most $T + \max_{j\in J} p_j$.

Solution:

We want to assign each job to a machine. For any job j which the LP assigns to a single machine i, i.e., $x^*_{ij} = 1$, it clearly makes sense to listen to the LP and assign j to i. To assign the other jobs, we will use the graph H, which has the following properties:

- H is acyclic, i.e., a forest.
- Vertices corresponding to the already-assigned jobs are isolated.
- Vertices corresponding to the yet-unassigned jobs have degree at least two.

While H contains an edge, we iteratively do the following:

- Choose a machine i whose vertex a_i has degree one in H, i.e., there is only a single edge $\{a_i, b_j\}$ incident to a_i. (Such a machine exists because H is nonempty, and a nonempty forest contains a degree-one vertex; since job-vertices have degrees either zero or at least two, a degree-one vertex must be a machine-vertex.)
- Assign job j to machine i.
- Remove all edges incident to j from H. (Note that this preserves all the above properties of H.)

Obviously, this procedure will terminate. Note that it will assign all jobs to machines, because as long as there is an unassigned job j, the vertex b_j has at least two incident edges in H.

It remains to reason that for every machine i, the sum of processing times of jobs assigned to i (this is called the *makespan* of i) is at most $T + \max_{j\in J} p_j$. This is because there are two kinds of jobs assigned to i:

- Assigned in the beginning: the jobs j which had $x^*_{ij} = 1$. The total processing time of these jobs is
$$\sum_{j\in J: x^*_{ij}=1} p_j = \sum_{j\in J: x^*_{ij}=1} x^*_{ij} p_j \le \sum_{j\in J: i\in N(j)} x^*_{ij} p_j \le T.$$
- Assigned later: at most one other job. For note that we only assign a job to i when a_i is of degree-one, and once we do, a_i becomes isolated and no more jobs can be assigned to i. This job has processing time at most $\max_{j\in J} p_j$.

Another perspective on the solution is the following. By looking at the makespan bound $T + \max_j p_j$ that we should satisfy, we can notice that it will be fine if we assign the integral jobs to the machines that the LP has selected and the non-integral jobs in such a way that each machine gets at most one. In other words, we just need to prove that the graph H has a matching which matches all non-integral jobs (any such matching will be fine). Our iterative procedure is one way to prove that such a matching exists.