CS 170
Spring 2017

Efficient Algorithms and Intractable Problems
Prasad Raghavendra and Sanjam Garg

Homework 8

Instructions: You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

If using LaTeX (which we recommend), you may use the homework template linked on this Piazza post to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the Homework FAQ Piazza post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 12 homework assignments, the lowest two scores will be dropped.

Special Questions:

- *Shortcut questions*: Short questions are usually easy questions that give you opportunities to practice basic materials. However, we understand that some of you are very familiar with the topics and do not want to spend too much time on easy questions. Therefore, we design shortcut questions for this purpose. A shortcut question usually has multiple parts that build upon each other and are ordered by their difficulty level. You can work on those in order or start from wherever you like. However you only need to submit the last part you are able to solve. For example, if a question has 5 parts (a, b, c, d, e), you are confident about part e, you should submit part e without any of the previous four parts. If you are confident about d but not sure about e, you should submit d for grading purposes. Please clearly indicate in your submission which part you are submitting.
- *Redemption questions*: It is important that you carefully read the posted solutions, even for problems you got right. To encourage this, you have the option of submitting a redemption file, a few paragraphs in which you explain, for each problem you choose to cover, what you did wrong and what the right idea was in your own words (not cutting and pasting from the solution!), and appending it to your homework. For example, suppose that as you review your solutions to HW1, you realize you had misunderstood question 3 and answered it incorrectly. You would write down what you just learned, and then submit it in your HW2 assignment the following week. Because these are mainly for your benefit, feel free to format them however is most useful for you.
- *Extra credit questions*: We might have some extra credit questions in the homework for people who really enjoy the materials. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your score.

Due Tuesday, April 11, at 11:59am

1. (★★ level) Salt

Salt is an extremely valuable commodity. There are m producers and n consumers, each with their own supply $[a_1 ... a_m]$ and demand $[b_1 ... b_n]$ of salt.

Note: Solve parts $(b), (c)$ independently of each other.

(a) Each producer can supply to any consumer they choose. Find an efficient algorithm to determine whether it is feasible for all demand to be met.

(b) Each producer is willing to deliver to consumers at most c_i distance away. Each producer i has a distance $d_{i,j}$ from consumer j. Solve part (a) with this additional constraint.

(c) Each producer and consumer now belongs to one of p different countries. Each country has a maximum limits on the amount of salt that can be imported (e_k) or exported (f_k). Deliveries within the same country don't contribute towards this limit. Solve part (a) with this additional constraint.

Solution:

(a) Formulate this as a max flow problem. Set up a bipartite graph, with one node for each of the m producers on the left, and one for each of the n consumers on the right.

For every producer i, create an edge between the source and producer of capacity a_i, to represent the maximum supply. Similarly, for every consumer j create an edge between the destination and consumer of capacity b_j to represent the maximum demand. Finally, create an edge between every producer and consumer of capacity ∞, since each producer can deliver to any consumers.

Running a max-flow algorithm on this graph, and checking whether the flow is greater than $\sum_j b_j$ will determine whether it is feasible for all demand to be met.

Alternate (Trivial) Solution: Simply check whether $\sum_i a_i \geq \sum_j b_j$. This answer was accepted for part (a), but doesn't provide a framework to solve parts (b), (c).

(b) We modify the max-flow solution from part a. Now only create the edge between a producer i and consumer j if the distance is within the limit; $d_{i,j} \leq c_i$.

Creating the intermediate edges in this manner ensures that producers can deliver only to consumers within their distance limit.

(c) We modify the max-flow solution from part a. For each country, create two dummy nodes X_k, X_k' and an edge between them of capacity e_k; this represents the limit on exports. Create dummy nodes Y_k, Y_k' and an edge between them of capacity f_k, to represent the limit on imports.

Now create an edge of capacity ∞ between each producer i and its respective export node X_k, as well as each consumer and its respective import node Y_k', to ensure that the import/exports are accounted for in the graph. Also, create an edge of capacity ∞ between any consumer and producer that are located in the same country, to represent deliveries within the same country. Finally, create an edge of capacity ∞ between every outgoing export node X_k' and incoming import node Y_k to allow trading between countries.

Funneling all exports/imports through country nodes X_k, Y_k' allows us to account for the total imports/exports for a country. The intermediate edges between the dummy nodes, $X_k \to X_k', Y_k \to Y_k'$ enforce the export/import limits e_k, f_k.

2. (★★★★ level) Minimum Cost Flows

In the max flow problem, we just wanted to see how much flow we could send between a source and a sink. But in general, we would like to model the fact that shipping flow takes money. More precisely, we are given a directed graph G with source s, sink t, costs l_e, capacities c_e, and a flow value F. We want to find a nonnegative flow f with minimum cost, that is $\sum_e l_e f_e$ that respects the capacities and ships F units of flow from s to t.

(a) Show how the minimum cost flow problem can be solved in polynomial time.

(b) Show how a minimum cost flow solver can solve shortest path problems.

(c) Show how a minimum cost flow solver can solve maximum flow problems.

Solution:

(a) Write min-cost flow as a linear program. One formulation is as follows:

$$\begin{aligned} &\min \sum_e l_e f_e \\ &\text{subject to } 0 \le f_e \le c_e && \forall e \in E \\ &\sum_{e \text{ incoming to } v} f_e = \sum_{e \text{ outgoing from } v} f_e && \forall v \in V, v \neq s,t \\ &\sum_{e \text{ outgoing from } s} f_e = F \end{aligned}$$

Note: this LP is different from the standard max-flow LP. For one, we are minimizing instead of maximizing, the objective is different, and we have the additional constraint that total flow = F at the end.

(b) Consider a shortest path instance on a graph G with start s, end t, and edge weights l_e. Let $F = 1$ and let $c_e = 1$ for all edges e.
The proof of correctness uses the fact that every flow can be decomposed into paths plus a circulation. Let f be the minimum cost flow, and $f = a_1P_1 + ... + a_kP_k + C$ be its decomposition into paths plus a circulation, with $a_1 + ... + a_k = 1$. If l is the length of the shortest path from s to t, then assume toward a contradiction that one of the P_j was of length more than l, then the cost of f would be at least $l + a_j * (|P_j| - l)$ which is more than l, but routing one unit along a shortest path would have cost l, which contradicts the optimality of f. So all P_i has a length of at most l.

(c) We can use binary search to find the true max flow value. Consider a max flow instance on a graph G with capacities c_e where we wish to ship flow from s to t. Create a min-cost flow instance as follows. Set the capacities to be equal to c_e and let the costs be arbitrary (say $l_e = 1$ for all edges). We know that the max flow value $F_{max} \le \sum_e c_e$. If the capacities are integers, F_{max} is also an integer. Therefore, we can binary search to find its true value. For an arbitrary F, we can find out if there is a flow that ships more than F units of flow by querying our min-cost flow instance with value at least F. If there is a flow with value at least F, it will return a flow with finite cost. Otherwise, the program is infeasible.

Note that this question requires a binary search approach because the formulation of min-cost flow requires finding a flow whose value is **exactly equal to F**. If the requirement were $\leq F$ or $\geq F$, then we will not have to do a binary search.

3. (★★★★ level) Minimum Spanning Trees

Consider the spanning tree problem, where we are given an undirected graph G with edge weights $w_{u,v}$ for every pair of vertices u, v.

An integer linear program that solves the minimum spanning tree problem is as follows:

$$\begin{aligned}
&\text{Minimize} \sum_{(u,v)\in E} w_{u,v}x_{u,v} \\
&\text{subject to} \sum_{\{u,v\}\in E: u\in L, v\in R} x_{u,v} \geq 1 \quad \text{for all partitions of } V \text{ into disjoint nonempty sets } L, R \\
&x_{u,v} \in \{0,1\}, \quad \forall (u,v) \in E
\end{aligned}$$

(a) Give an interpretation of the purpose of the objective function, decision variables, and constraints.

(b) Is creating the formulation a polynomial time algorithm with respect to the size of the input graph?

(c) Suppose that we relaxed the binary constraint on the decision variables $x_{u,v}$ with the non-negativity constraint:

$$x_{u,v} \geq 0, \quad \forall (u,v) \in E$$

How does the new linear program solution's objective value compare to the integer linear program's? Provide an example (decision variables and objective value) where the above LP relaxation achieves a better objective value than the ILP formulation.

Solution:

(a) $x_{u,v}$ is the binary decision variable corresponding to whether the edge (u,v) is used. The constraint $x_{u,v} \in \{0,1\}$ ensures that our decision variables $x_{u,v}$ correspond to a valid spanning tree; it would be unclear what $x_{u,v} = 0.5$ means in terms of the spanning tree.

The constraint $\sum_{(u,v)\in E'} x_{u,v} \geq 1$ ensures that there there is that there is at least one path in the solution for every possible cut in the graph, to ensure that the tree is a spanning tree.

The objective function minimizes the total cost of the tree; the weight of an edge $w_{u,v}$ is added to the cost if its corresponding decision variable $x_{u,v} = 1$.

(b) No; there are an exponential number of constraints to handle the exponential number of cuts.

(c) $v_{LP} \leq v_{ILP}$. The new linear program solution's objective value v_{LP} is at most integer linear program's objective value v_{ILP}, because the removal of constraints can only improve the objective value.

Assume for contradiction that $v_{LP} > v_{ILP}$. The solution set of decision variables for the ILP solution is guaranteed to be feasible for the LP formulation. We can therefore obtain v_{ILP} for the LP formulation, contradicting our assumption that v_{LP} was optimal for the LP formulation.

One example is a cycle with 3 nodes, $w_{u,v} = 1, \quad \forall u, v \in E$. The optimal ILP formulation picks any two of the edges for a total objective cost of 2. The optimal LP formulation picks $x_{u,v} = \frac{1}{2}$ for all edges, for a total objective cost of $\frac{3}{2}$.

4. (★★★ level) Major Key

You are a locksmith tasked with producing keys $k_1, \ldots, k_n$ that sell for $p_1, \ldots, p_n$ respectively. Each key k_i takes g_i grams of gold and s_i grams of silver. You have a total of G gold and S silver to work with, and can produce as many keys of any type as you want within the time and material constraints.

(a) Unfortunately, integer linear programming is an NP-complete problem. Fortunately, you have found someone to instead buy the alloys at an equivalent price! Instead of selling keys, you have decided to focus on melting the prerequisite metals together, and selling the mixture. Formulate the linear program to maximize the profit of the locksmith, and explain your decision variables, objective function, and constraints.

(b) Formulate the dual of the linear program from part (a), and explain your decision variables, objective function, and constraints. The explanations provide economic intuition behind the dual. We will only be grading the dual formulation.
Hint: Formulate the dual first, then think about it from the perspective of the locksmith when negotiating prices for buying G gold and S silver if they had already signed a contract for the prices for the output alloys p_i. Think about the breakeven point, from which the locksmith's operations begin to become profitable for at least one alloy.

Solution:

(a) The decision variables x_i correspond to the amount of mixture created for each key.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} x_i p_i && \text{(Maximize profit)} \\ & \sum_{i=1}^{n} x_i g_i \leq G && \text{(Use at most } G \text{ grams of gold)} \\ & \sum_{i=1}^{n} x_i s_i \leq S && \text{(Use at most } S \text{ grams of silver)} \\ x_i \geq 0 \quad & \forall i \in [1...n] && \text{(Cannot produce negative amounts of metal)} \end{aligned}$$

(b) The decision variables y_G, y_S correspond to the prices of the means of production: gold, and silver. The dual poses the following question: if the prices of gold and silver were originally too high for the locksmith's operations to be profitable, how low can they drop before breaking even? The solution returns the breakeven point; lower prices would finally allow the locksmith to become profitable.

$$\begin{aligned} \min \quad & G y_G + S y_S && \text{(Minimize total cost of materials)} \\ g_i y_G + s_i y_S \geq p_i \quad & \forall i \in [1...n] && \text{(Cost for producing mixture } i \text{ is at least } p_i) \\ & y_G, y_S \geq 0 && \text{(Cannot set negative prices)} \end{aligned}$$

5. (★★★ level) Zero-Sum Battle

Two Pokemon trainers are about to engage in battle! Each trainer has 3 Pokemon, each of a single, unique type. They each must choose which Pokemon to send out first. Of course each trainer's advantage in battle depends not only on their own Pokemon, but on which Pokemon their opponent sends out.

The table below indicates the competitive advantage (payoff) Trainer A would gain (and Trainer B would lose). For example, if Trainer B chooses the fire Pokemon and Trainer A chooses the rock Pokemon, Trainer A would have payoff 2.

		Trainer B: ice	grass	fire
Trainer A:	dragon	-10	3	3
	steel	4	-1	-3
	rock	6	-9	2

Feel free to use an online LP solver to solve your LPs in this problem.
Here is an example of a Python LP Solver and its Tutorial.

(a) Write an LP to find the optimal strategy for Trainer A. What is the optimal strategy and expected payoff?

Solution:
d = probability that A picks the dragon type
s = probability that A picks the steel type
r = probability that A picks the rock type

$$\begin{aligned} \max \quad & z \\ -10d + 4s + 6r &\geq z && \text{(B chooses ice)} \\ 3d - s - 9r &\geq z && \text{(B chooses grass)} \\ 3d - 3s + 2r &\geq z && \text{(B chooses fire)} \\ d + s + r &= 1 \\ d, s, r &\geq 0 \end{aligned}$$

The optimal strategy is $d = 0.3346,\ s = 0.5630,\ r = 0.1024$ for an optimal payoff of -0.48.

(b) Now do the same for Trainer B. What is the optimal strategy and expected payoff?

Solution:
i = probability that B picks the ice type
g = probability that B picks the grass type
f = probability that B picks the fire type

$$\begin{aligned} \min \quad & z \\ -10i + 3g + 3f &\leq z && \text{(A chooses dragon)} \\ 4i - g - 3f &\leq z && \text{(A chooses steel)} \\ 6i - 9g + 2f &\leq z && \text{(A chooses rock)} \\ i + g + f &= 1 \\ i, g, f &\geq 0 \end{aligned}$$

B's optimal strategy is $i = 0.2677,\ g = 0.3228,\ f = 0.4094$. The value for this is -0.48, which is the payoff for A. The payoff for B is 0.48, since the game is zero-sum.

(Note for grading: Equivalent LPs are of course fine. It is fine for part (b) to maximize B's payoff instead of minimizing A's. For the strategies, fractions or decimals close to the solutions are fine, as long as the LP is correct.)

6. (★★★ level) Decision vs. Search vs. Optimization

The following are three formulations of the VERTEX COVER problem:

- As a *decision problem*: Given a graph G, return TRUE if it has a vertex cover of size at most b, and FALSE otherwise.
- As a *search problem*: Given a graph G, find a vertex cover of size at most b (that is, return the actual vertices), or report that none exists.
- As an *optimization problem*: Given a graph G, find a minimum vertex cover.

At first glance, it may seem that search should be harder than decision, and that optimization should be even harder. We will show that if any one can be solved in polynomial time, so can the others:

Describe your algorithms precisely; justify correctness and running time. No pseudocode.

Hint for both parts: Call the black box more than once.

(a) Suppose you are handed a black box that solves VERTEX COVER (DECISION) in polynomial time. Give an algorithm that solves VERTEX COVER (SEARCH) in polynomial time.

(b) Similarly, suppose we know how to solve VERTEX COVER (SEARCH) in polynomial time. Give an algorithm that solves VERTEX COVER (OPTIMIZATION) in polynomial time.

Solution:

(a) If given a graph G and budget b, we first run the DECISION algorithm on instance (G, b). If it returns "FALSE", then report "no solution".

If it comes up "TRUE", then there is a solution and we find it as follows:

- Pick any node $v \in G$ and remove it, along with any incident edges.
- Run DECISION on the instance $(G \setminus \{v\}, b-1)$; if it says "TRUE", add v to the vertex cover. Otherwise, put v and its edges back into G.
- Repeat until G is empty.

Correctness: If there is no solution, obviously we report as such. If there is, then our algorithm tests individual nodes to see if they are in any vertex cover of size b (there may be multiple). If and only if it is, the subgraph $G \setminus \{v\}$ must have a vertex cover no larger than $b-1$. Apply this argument inductively.

Running time: We may test each vertex once before finding a v that is part of the b-vertex cover and recursing. Thus we call the DECISION procedure $O(n^2)$ times. This can be tightened to $O(n)$ by not considering any vertex twice. Since a call to DECISION costs polynomial time, we have polynomial complexity overall.

Note: this reduction can be thought of as a greedy algorithm, in which we discover (or eliminate) one vertex at a time.

(b) Binary search on the size, b, of the vertex cover. For example, if running DECISION on an instance (G, b) gives no solution, then try again with $(G, 2b)$. If it gives a solution, then try again with $(G, b/2)$, etc. Proceed until the minimum viable b is found.

Correctness: This algorithm is correct for the same reason as binary search.

Running time: The minimum vertex cover is certainly of size at least 1 (for a nonempty graph) and at most $|V|$, so the SEARCH black box will be called $O(\log |V|)$ times, giving polynomial complexity overall.

Finally, since solving the optimization problem allows us to answer the decision problem (think about why), we see that all three reduce to one another!

Note that the reductions here are slightly different from what we will typically use because we are allowed to query the oracle (black box) multiple times here. As a result we have not actually shown the optimization problem to be in **NP**. Instead, we can only say that it is in a (possibly) larger complexity class known as $\mathbf{P}^{\mathbf{NP}}$. This is slightly beyond the scope of this course.