CS 170 Fall 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 Wednesday, November 8, at 4:59pm

0. Who did you work with?

List all your collaborators on this homework. If you have no collaborators, please list “none”.

Solution: N/A

1. **(★★★ level) iPhone X**
Apple has just gotten hacked and their instruction manual for producing their new iPhone X has been released to the public. A couple factories that specialize in producing knockoff popular consumer products take advantage of this and start producing their own cheaper versions of the iPhone X. These factories do not have the means for distribution, but fortunately, a few companies that specialize in distribution of knockoff products are interested in this promising business opportunity. There are a total of m factories, each capable of producing a_i knockoff iPhones. There are also n distributors, each demanding b_i knockoff iPhones.

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

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

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

(c) Each factory and distributor now belongs to one of p different countries. Each country has a maximum limit on the amount of iPhones 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 factories on the left, and one for each of the n distributors on the right.

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

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 factory i and distributor j if the distance is within the limit; $d_{i,j} \leq c_i$.

Creating the intermediate edges in this manner ensures that factories can deliver only to distributors 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 factory i and its respective export node X_k, as well as each distributor 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 distributor and factory 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) What happens in Vegas...

The Venetian at Las Vegas has just introduced a brand new game, Rambis. The rules are simple. You and the dealer both have three cards - Jack, Queen, and King - and you both place one on the table face down. Then the dealer will flip both the cards and pay you a certain amount based on the table below.

Note that a negative value means you owe the casino that amount. For example, if you chose Jack and the dealer also chose Jack, you would owe the casino \$10.

		Dealer:		
		Jack	Queen	King
	Jack	-10	3	3
You:	Queen	4	-1	-3
	King	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 your optimal strategy. What is the optimal strategy and expected payoff?

Solution:
d = probability that you pick Jack
s = probability that you pick Queen
r = probability that you pick King

$$\begin{aligned} &\max \quad z \\ -10d+4s+6r &\geq z && \text{(dealer chooses Jack)} \\ 3d-s-9r &\geq z && \text{(dealer chooses Queen)} \\ 3d-3s+2r &\geq z && \text{(dealer chooses King)} \\ d+s+r &= 1 \\ d,s,r &\geq 0 \end{aligned}$$

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 the dealer. What is the optimal strategy and expected payoff?

Solution:
i = probability that dealer picks Jack
g = probability that dealer picks Queen
f = probability that dealer picks King

$$\begin{aligned} &\min \quad z \\ -10i+3g+3f &\leq z && \text{(you choose Jack)} \\ 4i-g-3f &\leq z && \text{(you choose Queen)} \\ 6i-9g+2f &\leq z && \text{(you choose King)} \\ i+g+f &= 1 \\ i,g,f &\geq 0 \end{aligned}$$

B's optimal strategy is $i = 0.2677,\ g = 0.3228,\ f = 0.4094$. The value for this is -0.48, which is your payoff. The payoff for the casino 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 the dealer's payoff instead of minimizing yours. For the strategies, fractions or decimals close to the solutions are fine, as long as the LP is correct.)

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

Consider the spanning tree problem, where we are given an undirected and connected graph $G : (V, E)$ 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:

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

(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) 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.

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.

5. (★★★★ level) Max-Flow Variants

Show how to reduce the following variants of Max-Flow to the regular Max-Flow problem, i.e. do the following steps for each variant: Given a graph G and the additional variant constraints, show how to construct a graph G' such that

(1) If F is a flow in G satisfying the additional constraints, there is a flow F' in G' of the same size,

(2) If F' is a flow in G', then there is a flow F in G satisfying the additional constraints with the same size.

Prove that properties (1) and (2) hold for your graph G'.

(a) **Max-Flow with Vertex Capacities:** In addition to edge capacities, every vertex $v \in G$ has a capacity c_v, and the flow must satisfy $\forall v : \sum_{u:(u,v)\in E} f_{uv} \leq c_v$.

(b) **Max-Flow with Multiple Sources:** There are multiple source nodes $s_1, \dots, s_k$, and the goal is to maximize the total flow coming out of all of these sources.

(c) **Feasibility with Capacity Lower Bounds:** In addition to edge capacities, every edge (u,v) has a demand d_{uv}, and the flow along that edge must be at least d_{uv}. Instead of proving (1) and (2), design a graph G' and a number D such that if the maximum flow in G' is at least D, then there exists a flow in G satisfying $\forall (u,v) : d_{uv} \leq f_{uv} \leq c_{uv}$.

Solution:

(a) It suffices to split every vertex into an 'incoming' and 'outgoing' vertex, and connect the two via an edge whose capacity is the capacity of the original vertex. This "internal" edge enforces the vertex capacity constraint, because all flow that goes into and out of the original vertex will be funneled through this edge. Details follow.

Split every vertex v into two vertices, v_{in} and v_{out}, and direct all incoming edges for v into v_{in} and all outgoing edges out of v_{out} as follows: for each edge (u,v) with capacity c_{uv} in the original graph, create an edge (u_{out}, v_{in}) with capacity c_{uv}. Finally, if v has capacity c_v, then create an edge (v_{in}, v_{out}) with capacity c_v. If F' is a flow in this graph, then setting $F(u,v) = F'(u_{out}, v_{in})$ gives a flow in the original graph. Moreover, since the only outgoing edge from v_{in} is (v_{in}, v_{out}), and incoming flow must be equal to outgoing flow, there can be at most c_v flow passing through v. Likewise, if F is a flow in the original graph, setting $F'(u_{out}, v_{in}) = F(u,v)$, and $F'(v_{in}, v_{out}) = \sum_u F(u,v)$ gives a flow in G'. One can easily see that these flows have the same size.

(b) A source can put out as much flow as can be carried by later edges, so there is no harm to add a meta-source that connects to all of these sources via edges having infinite capacity. Details follow.

Create one "supersource" S with edges (S, s_i) for each s_i, and set the capacity of these edges to be infinite. Then if F is a flow in G, set $F'(S, s_i) = \sum_u F(s_i, u)$. Conversely, if F' is a flow in G', just set $F(u,v) = F'(u,v)$ for $u \neq S$, and just forget about the edges from S. One can easily see that these flows have the same size.

(c) The intuitive solution is to simply subtract off the demands from the capacities, and run max flow. However, this is not enough because a valid flow on G' does not correspond to a valid flow on G, because when you add back the demands you may violate some flow conservation constraints (e.g., try to take the zero flow from G' to G). However, this simple idea is in the right direction, because there is a way to modify G' so that its valid flows do correspond to valid flows on G. Roughly, the idea is to add, for each vertex v, one incoming edge and one outgoing edge that totals the demands of the incoming and outgoing edges in the original graph, then subtract the demands from the original capacities of the edges. The new incoming and outgoing edge would carry the flow that corresponds to the demand; because this flow must be conserved, when moving back to G, we do get a valid flow after adding back the demands. Details follow.

Add two vertices to G, call them s' and t', and add edges (s',v) with capacity $\sum_u d(u,v)$ and (v,t') with capacity $\sum_u d(v,u)$. Add an edge (t,s) with capacity ∞, and change the capacities of all the edges in G to $c(u,v)-d(u,v)$. Let $D=\sum_{(u,v)\in E} d(u,v)$. We consider s' and t' to be the new source and sink. Note that the cuts that consist of only $\{s'\}$ or only $\{t'\}$ have value D, so D is an upper bound on the size of the max-flow.

If G has a feasible flow F, we construct a flow F' on G' by fully saturating the edges leaving s' and coming into t', setting $F'(u,v)=F(u,v)-d(u,v)$, and $F'(t,s)=\text{size}(F)$. Since $F(u,v)\leq c(u,v)$, all the capacity constraints are satisfied, and adding up the incoming flow at a single vertex, we get

$$\sum_{u\in G'} F'(u,v) = F'(s',v) + \sum_{u\in G}(F(u,v)-d(u,v)) = \sum_{u\in G} F(u,v)$$

which is the incoming flow to v in F. Likewise, the outgoing flow in F' is also equal to the outgoing flow in F, and since F is a flow these must be equal, so indeed F' is a valid flow. Since all the (s',v) edges are fully saturated, this flow has size exactly D.

To show the other direction, let G' have a flow of size exactly D. Note this implies that all the (s',v) and (v,t') edges must be fully saturated, since the corresponding cuts have value D. Set $F(u,v)=F'(u,v)+d(u,v)$. Since $F'(u,v)\leq c(u,v)-d(u,v)$, these values satisfy the capacity and demand constraints. Now adding up the incoming flow at a single (non-source or -sink) vertex,

$$\sum_{u\in G} F(u,v) = \sum_{u\in G', u\neq s'}(F'(u,v)+d(u,v)) = \sum_{u\in G', u\neq s'} F(u,v) + F(s',v) = \sum_{u\in G'} F'(u,v)$$

where the last equality holds because each $F(s',v)$ must be fully saturated. Thus the flow incoming into v in F is equal to the flow incoming into v in F'. A similar argument shows the outgoing flow from v in F is equal to the outgoing flow from v in F'. Since F' is a flow, these must be equal, and so F is a flow.

6. (??? **level) (Optional) Redemption for Homework 7**

Submit your *redemption file* for Homework 7 on Gradescope. If you looked at the solutions and took notes on what you learned or what you got wrong, include them in the redemption file.

Solution: N/A