CS 170: Efficient Algorithms and Intractable Problems

Spring 2017

•

Homework 8

Due on Tuesday, April 11th, 2017 at 11:59am

•

Solutions by

Michael Fan

26697596

In collaboration with

Alex Gao, Julie Han

Problem 1: Salt

★★ **Level**

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

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

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

Solution Check that sum of supply $a_1 ... a_m$ is greater than or equal to the sum of demand $b_1 ... b_n$.

(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.

Solution We will formulate this as a max flow problem. All producers and consumers are represented as nodes. Each producer has an edge to a consumer if the distance constraint is met. We will create source node s that has directed edges to all producers - these m edges have the same capacity as the supply a_m of those m producers. The edges from the producers to consumers have capacity equivalent to the consumers' demand b_n, and the n consumers' edges to the sink have combined capacity b_n.
We now run the max flow algorithm on the constructed graph. We check if the max flow is equivalent to the sum of the demands given by the b_n units of flow passing through to the consumer nodes and to the sink node. If the max flow is less than b_n, that means there are consumer(s) who did not have their demand met.

(c) Each producer and consumer now belongs to one of the p different countries. Each country has a maximum limit 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 We formulate this as an LP problem. The objective function is $(x_1 + x_2 + ... + x_{mn})$, where x_{ij} represents an amount of flow from producer i to consumer j. The constraints follow:

$$\forall m, (\sum x_{mn} \leq a_m)$$

$$\forall b(\sum x_{mn} \leq e_k) \text{for n in b, m not in b}$$

$$\forall b(\sum x_{mn} \leq f_k) \text{for m in b, n not in b}$$

The first set of constraints guarantee that for all m suppliers, no one ships out more supply than they had to begin with.
The second set of constraints represents that imports into a country p from suppliers who are not in that country do not exceed the import limit for that country.
The third set of constraint represents that exports out of a country p to consumers who aren't in that country do not exceed the export limit.

Problem 2: Minimum Cost Flows

★★★★ Level

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.

Solution The min cost flow problem can be formulated with LP, which is solved in polynomial time with simplex.

Objective function:

$$min \sum l_e * f_e$$

Constraints:

$$\sum f_e \text{ , e is edge leading to sink} = F$$

$$f_e \leq c_e$$

$$\text{for all nodes}(\sum f_e, \text{ e is edge going into a node} - \sum f_e \text{ , e is edge going out of node} \geq 0)$$

Constraint 1 follows as the flows need to add up to the desired flow.
Constraint 2 follows as we don't want flow on any edge to exceed the capacity of the edge.
Constraint 3 follows as we don't want flow coming out of a node to exceed flow going into it.

(b) Show how a solver for the minimum cost flow problem can also generate solutions for the shortest path problem.

Solution Set the constraint on the flow to equal 1, and set the cost of each edge equal to any constant c. The minimum cost to ship 1 flow through the graph corresponds to the path which has the fewest edges. We prove this via contradiction. Assume there exists a shortest path s. Assume that there is a second path t that is also min-cost but contains more edges than s. In that case, the cost of this path f can be improved by switching to path s, since the total cost $c|f| > c|s|$, which is a contradiction.

(c) Show how a solver for the minimum cost flow problem can also generate solutions for the maximum flow problem.

Solution We know that we can just reuse the minimum cost solver over and over with arbitrary values for F. Thus, we can use a binary search in which we test some arbitrary F to see if a min-cost flow exists, then test $2F, 3F, ...$ until we have reached some multiple of F such that the LP is infeasible. Then you go back downwards in increments smaller than F until it's feasible, then go back upwards in even smaller increments till it's infeasible. When the binary search terminates, we will have found F_0 that is the max flow that allows the problem to be feasible.

Problem 3: Minimum Spanning Trees

★★★★ Level

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:

Minimize

$$\sum_{(u,v)\in E} w_{u,v}x_{u,v}$$

subject to

$$\sum_{\{u,v\}\in E:u\in L,v\in R} x_{u,v} \geq 1$$

for all partitions of V into disjoint nonempty sets L, R

$$x_{u,v} \in \{0,1\}, \quad \forall (u,v) \in E$$

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

Solution The objective function minimizes the sum of all the edges. The x is an indicator variable that tells whether or not to add an edge to the sum. Each w, the weight of an edge, is multiplied by an x to indicate if it is included in the MST.

The first constraint indicates that there must exist at least one edge across any cut. This guarantees the connectivity of an MST: if there exists two disjoint sets with no edges between them at all, then there is not a spanning tree.

The second constraint means that all edges have to be of integer multiplicity as there is no meaning behind fractional edges.

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

Solution No. Each additional vertex in the input graph doubles the amount of constraints. The original n L/R partitions described in the first set of constraints can now include the new vertex in either L or R, meaning there are now $2n$ partitions and double the amount of constraints. Thus, formulating the problem is exponential in 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's 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 Given an equilateral triangle of side length 1, regular MST would have an objective value of 2. The Relaxed LP MST solver we have would weight each edge by $x = 1/2$ to achieve an objective value of 1.5 while still maintaining that each cut has edges crossing it which have x values summing to 1.

Problem 4: Major Key

★★★ Level

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.

Solution Objective function:

$$max \sum_{i=1}^{n} x_i * p_i$$

where p_i is the price of that particular alloy/key i, and x_i is how much of the particular key i is made. This is basically just maximizing profit.
Constraints:

$$\sum_{i=1}^{n} x_i * g_i \leq G$$

$$\sum_{i=1}^{n} x_i * s_i \leq S$$

Constraint 1 indicates that the total amount of gold used to make all the amounts of alloys cannot exceed your starting supply
Constraint 2 indicates that the total amount of silver cannot exceeding starting supply.

(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 Objective function:

$$miny_1 * G + y_2 * S$$

. Note that we are now minimizing costs. Follow the constraints (represented in matrix form)

$$y^T * A \geq p$$

where y is the vector of decision variables (y_1 and y_2), A is a matrix formed of n vectors of height 2 which represent the amount of gold/silver necessary to make a particular key i, and p is the vector $p_1 ... p_n$ of prices that each key sells for.

Explanation for objective function: the locksmith wants to minimize the unit price he pays for the fixed amounts of gold and silver. Explanation for constraint function: the cost that he must incur for the raw materials for an alloy (a column in A) must also be more than the revenue gained from selling that alloy (the corresponding entry in p).

Problem 5: Zero-Sum Battle

★★★ **Level**

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 A chooses the fire Pokemon and Trainer B 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
$maxz$
$x1 + x2 + x3 == 1$
$10 * x1 - 4 * x2 - 6 * x3 + z <= 0$
$-3 * x1 + x2 + 9 * x3 + z <= 0$
$-3 * x1 + 3 * x2 - 2 * x3 + z <= 0$
Solved with PuLP
$x1 = 0.334646$
$x2 = 0.562992$
$x3 = 0.102362$
$z = -0.480315$
Pick dragon 0.34 of the time, steel 0.56 of the time, rock 0.1 of the time for an expected payoff of -0.48.

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

Solution
$minw$
$y1 + y2 + y3 == 1$
$10 * y1 - 3 * y2 - 3 * y3 - w <= 0$
$-4 * y1 + y2 + 3 * y3 - w <= 0$
$-6 * y1 + 9 * y2 - 2 * y3 - w <= 0$
$w = 0.480315$
$y1 = 0.267717$
$y2 = 0.322835$
$y3 = 0.409449$
Pick ice 0.27 of the time, grass 0.32 of the time, fire 0.41 of the time for an expected payoff of 0.48.

Problem 6: Decision vs. Search vs. Optimization

★★★ Level

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, up to polynomial factors, they actually have the same difficulty:
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.

Solution One can iterate through all vertices and check for each vertex if running decision box on the graph with that vertex removed returns FALSE. If it returns FALSE, then add the removed vertex back. If the decision box returns TRUE, do not add the vertex back and go to the next vertex. If you have removed enough vertices such that the remaining graph contains b vertices, then return that set. Otherwise, if removing any vertex and running decision box returns FALSE before you are down to b vertices, return that no such set exists.
This algorithm is correct because we will have trimmed all unnecessary vertices for one distinct vertex cover of size b. Even if there are multiple sets of such size, we ensure the integrity of the set that we are left with by removing vertices one at a time and adding them back if removing it leads to a situation in which no complete vertex cover of size b remains. This is polynomial time because decision box is polynomial, and you're running it exactly $|V|$ times. Therefore, the runtime of this algorithm is the same runtime as decision box but scaled up by $|V|$ which is still polynomial.

(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 The optimal solution has an upper bound of $|V|$ vertices (every vertex is needed). Run binary search on the number of vertices you include (bounded between 0 and $|V|$). Use the current number in the search as input b for the search box. If the search box returns that there is no such set, pick the larger half of the range to recurse binary search on. When the search box returns that there is such a set, recurse on the smaller half of the range. When binary search halts, return the last valid vertex cover.
This algorithm is correct because we will have checked all possible numbers n for which there could be a valid set cover of size n by expanding our range when there is not a valid vertex cover of size n.
This is polynomial because we essentially plugged search box into a binary search with range $(0, |V|)$ which should terminate in $\log|V|$ iterations. So, the runtime is the runtime of search box which is polynomial multiplied by $O(log|V|)$ which is still polynomial.