CS 170: Efficient Algorithms and Intractable Problems

Spring 2017

•

Homework 8

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

•

Solutions by

Ninh DO

25949105

In collaboration with

NONE

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

Main Idea:

We use max-flow algorithm.

Consider two sets: producers P and consumers C, we set up a source S directing to each producer P_i with edge of capacity a_i, a sink T to which each consumer C_i directs with edge of capacity b_i. Then we set edges of capacity ∞ directing from each producer to every consumer.

We run the max-flow algorithm for the above graph.

Proof of Correctness:

It is correct because we use the well-known max-flow alrogrithm. The edges a_i upper-bound the producers' supply, the edges a_i upper-bound the consumers' demand. The infinity edges between P and C allow one producer can provide salt to multiple consumers and one consumer can receive salt from multiple producers.

Runtime:

The runtime is $O\left((|P| + |C| + |E|) * |E|\right)$, where $|E| = \Theta\left(|P| * |V|\right)$ is the number of edges. Thus the worst case runtime is $O\left(|P|^2 * |C|^2\right)$

(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

Main Idea:

We modify the above graph in the way that we only set the infinity edges ∞ from P_i to C_i if their distance $d_{i,j} \leq c_i$. Then we run the max-flow algorithm for the modified graph.

Proof of Correctness:

Similar to part (a), since we only set edges between producer and consumer of distance within the limit, we eliminate the possibility that the producer delivers salt to the consumer out of the limit distances.

Runtime:

The runtime is $O\left(|E|^2\right)$, where $|E|$ is the number of edges between P and C.

(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

Main Idea:

We modify the original graph in the following way.

First, we duplicate each country i in the set of producers' countries into E_{1_i} and E_{2_i}, and we duplicate each country j in the set of consumers' countries into I_{1_j} and I_{2_j}.

We connect E_{1_i} to E_{2_i} by the directed edge of capacity f_i representing the export limit, and I_{1_i} to I_{2_i} by the directed edge of capacity e_i representing the import limit. We connect each E_{2_i} to every I_{1_j} by the edges of capacity ∞

Solution (cont.)

We connect the producers to their corresponsing countries E_{1_i}, and the countries I_{j_2} to the consumers, all by the edges of capacity ∞.
We connect the producer and consumer of the same country by the edge of infinity ∞.
Finally, we run the max-flow algorithm for the modified graph.
Proof of Correctness:
Similar the the previous parts, the bottle necks between E_{1_i} and E_{2_i}, I_{1_j} and I_{2_j} represent the export and import limits, respectively. **Runtime:**
The runtime is $O\left(|E|^2\right)$ where $|E|$ is the number of edges between export and import countries, because it dominate the other number of edges.

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 new problem is formulated into:

$$\begin{aligned}
\min \quad & \sum_e l_e f_e \\
\text{subject to} \quad & f_e \le c_e \\
& \sum_{\text{into a vertex}} f_e = \sum_{\text{out of that vertex}} f_e \qquad \text{for all vertices not source and sink} \\
& \sum_{\text{out of S}} f_e = F \\
& f_e \ge 0
\end{aligned}$$

Since this is a max-flow problem so it can be solve in polinomial time.

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

Solution

Consider a shortest parth from s to t on the graph G with F = 1 and all $c_e = 1$, if we can show that in the subgraph G' with the min-cost flow f and all edges of nonzero flow, all directed path from s to t have length equal to the shortest path from s to t, then we can use DFS or BFS to find the shortest path from s to t.

First, G' is a DAG. If G' is not, then there is a minimum flow a long an edge of a cycle of G'. If we substract this flow from the other flows in the cycle, it results in a smaller flow than the min-cost flow f. This is contradiction, so G' is a DAG.

Let δ be the minimum flow on any path between s and t in G'. We can write:

$f = \delta p + \delta_1 p_1 + ... + \delta_i p_i$

where p and pi are the unit flows along the paths P and Pi We can decompose the cost and decrease the cost of f.

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

Solution

We can use binary search.

Step 1. Set all capacities to c_e and arbitrary cost

Step 2. If the capacities are integers then F max is an integer, thus we use binary search to find the true value

If F is arbitrary, we can solve the min-cost flow problem with at least F. If there is such a flow, then the finite cost is found, otherwise it cannot be solved.

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 is the total weights of selected edges based on the decision variables. If the decision variable is 1, the corresponding edge weight is taken into the objective function. If the decision variable is 0, it is not.
The decision variables indicate which edges belong to the minimum spanning tree, or in other words, which edges are selected.
The constraints are to make sure that there is at least one edge across an arbitrary partition/cut, that is to make sure that the graph is still connected after the selection of edges for the minimum spanning tree.

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

Solution
NO. It should be exponential time since we have to consider all partitions of a vertex set, that is any kind of combinations of any number of vertices.

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

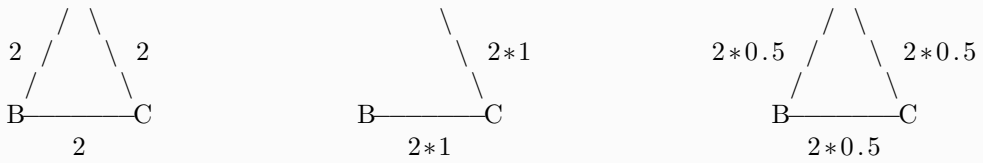
$$x_{u,v} \geq 0, \quad \forall (u,v) \in E$$

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
The new linear program solution's objective value should be lower than or equal to the integer linear program's. Since the binary constraint is relaxed, i.e. the new contraint covers the old constraint, the new objective value is at worst equal to the old objective value. If we can indicate at least one case that the new objective value is lower than the old objective value, the above statement is correct. Let's consider the following graph:

Original graph	ILP (MST)	LP Relaxation
A	A	A

Solution (cont.)



where, the second factor of each weight is the selected decision variable.
The optimal value of ILP is 4 while that of LP Relaxation is 3.

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

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} x_i p_i \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i g_i \leq G \\ & \sum_{i=1}^{n} x_i s_i \leq S \\ & x_i \geq 0, \quad 1 \leq i \leq n \end{aligned}$$

Similar to the integer programming problem (ILP), we replace the integer number of keys k_i of each type with the non-integer 'number of keys' x_i of each type. The decision variables x_i can be considered weights of alloys with components (g_i, s_i)
Each alloy i has weight x_i and price p_i, so the objective function is to maximize the total value of all alloys taking their weights into account.
For all alloys, the constraints make sure that their total weights of gold and silver components, g_i, s_i respectively, must be upper-bounded by G and S respectively.

(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

$$\begin{aligned} \min \quad & \alpha G + \beta S \\ \text{subject to} \quad & \alpha g_i + \beta s_i \geq p_i, \quad 1 \leq i \leq n \\ & \alpha, \beta \geq 0 \end{aligned}$$

The decision variable α and β can be considered the gold and silver prices, respectively, per unit. The objective function that we try to minimize is the amount of money we pay for G gold and S silver, subject to the constraints that the values of alloys i must be at least the predetermined prices p_i, respectively, which is the breakeven point.

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 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
Pick (x_1, x_2, x_3) that maximizes $\min\{-10x_1 + 4x_2 + 6x_3, 3x_1 - x_2 - 9x_3, 3x_1 - 3x_2 + 2x_3\}$

$$\begin{aligned} \max \quad & z \\ \text{subject to} \quad & 10x_1 - 4x_2 - 6x_3 + z \leq 0 \\ & -3x_1 + x_2 + 9x_3 + z \leq 0 \\ & -3x_1 + 3x_2 - 2x_3 + z \leq 0 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

$x_1 = 0.334646$
$x_2 = 0.562992$
$x_3 = 0.102362$
z = -0.480315

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

Solution
Pick (y_1, y_2, y_3) that minimizes $\max\{-10y_1 + 3y_2 + 3y_3, 4y_1 - y_2 - 3y_3, 6y_1 - 9y_2 + 2y_3\}$

$$\begin{aligned} \min \quad & w \\ \text{subject to} \quad & 10y_1 - 3y_2 - 3y_3 + w \geq 0 \\ & -4y_1 + y_2 + 3y_3 + w \geq 0 \\ & -6y_1 + 9y_2 - 2y_3 + w \geq 0 \\ & y_1 + y_2 + y_3 = 1 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

$y_1 = 0.267717$
$y_1 = 0.322835$
$y_1 = 0.409449$
w=-0.480315

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
Main Idea:
First, we run the black box on G with b. If it returns FALSE, then there is no solution. Otherwise, we run the following algorithm:

(a) Run the black box on $(G - \{v\}, b - 1)$, where v is a chosen vertex. If it return TRUE, add v to the vertex cover. Otherwise, put v back into G.

(b) Repeat 1. until G is empty.

Proof of Correctness:
For each vertex, the black box returns if it is in a vertex cover, working on a sub-problem $(G-\{v\}, b-1)$. This is repeated again and again until the graph G is empty, so it will tell the vertex cover of size at most b.
Runtime:
We call the black box $O(|V|)$ times at best and $O(|V|^2)$ times at worst. The black box runs in polynomial time, so our algorithm also run 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
Main Idea:
We use midpoint method. We call the black box on (G, b), if it returns FALSE, then call the black box on $(G, 2b)$, otherwise on $(G, b/2)$. If the calling black box on $(G, 2b)$ returns TRUE, then call the black box on $(G, b + b/2)$, otherwise on $(G, 4b)$ etc.
Proof of Correctness:
Since the black box and the midpoint methods is correct, our algorithm using both methods is correct.
Runtime:
We call the black box $O(\log |V|)$ times. The black box runs in polynomial time, so our algorithm also run in polynomial time.