15-451/651 Assignment 5

Vy Nguyen
vyn
Recitation: B
April 3, 2018

1: Safety First!

(a)
We will show that the problem is NP-hard via reducing Independent Set $\leq_P$ Uranium$_a$. Consider an arbitrary instance I of Independent Set, in which the inputs are

1. integer m
2. an undirected graph $G = (V, E)$ where $|V| = n, |E| = m$

Let the inputs of the Uranium$_a$ problem be

1. integer $D = 2$
2. integer m',
3. an undirected graph $G' = (V', E')$

For any instance I, we can map it to an instance $f(I)$ of the Uranium$_a$ problem. The mapping $f : I \to f(I)$ is defined by letting $V' = V, E' = E, m' = m$ and assign each edge in G' with a weight of 1. Since it takes $O(\log n)$ bits to specify the vertices, $O(m \log n) + O(m \log(\text{max flow}))$ $= O(m \log n)$ bits to specify the edges, and $\log n$ bits each to specify D and m', the mapping f runs in polynomial time of its input size G, namely $O(m \log n)$.

Now suppose that we have access to two black box algorithms for each problem. The algorithm for Independent Set returns YES when G has an independent set of size of at least m and NO otherwise. Similarly, the algorithm for Uranium$_a$ returns YES when we can place at least m' units of uranium in G' safely and NO otherwise.

Consider the decisional versions of the problems. We need to show that
G has an independent set of size at least m
$\Leftrightarrow$ G' has at least m' units of uranium without triggering an explosion.

($\Rightarrow$) Suppose there exists an independent set of size at least m in G and the Independent Set algorithm returns YES. Since the algorithm for Independent Set would not pick two neighboring vertices, this is the same as not picking two storage facilities whose distance is less than D. This corresponds to the set of size at least $m' = m$ viable facilities to place uranium in without triggering an explosion, a YES instance in the Uranium$_a$ problem.

($\Leftarrow$) Suppose we have a YES instance in the Uranium$_a$ problem and can choose a set of at least m' vertices in G' to safely place the uranium. Then for any two of these chosen vertices, their distance must be greater than 2. Thus, these vertices cannot be neighbors to each other since by construction, each edge in G' has a weight of 1, i.e. the vertices form an independent set of size at least $m' = m$. This corresponds to a YES instance to the Independent Set problem.

(b)
We will show that the problem is NP-hard via reducing Set Cover $\leq_P$ Uranium$_b$. Consider an arbitrary instance I of Set Cover, in which the inputs are

1. an integer k
2. a set of vertices $U = \{v_1, ..., v_n\}$
3. a collection of subsets of U, $S = \{S_1, ..., S_m\}$

Let the inputs of the Uranium$_b$ problem be

1. an integer k'
2. an undirected graph $G = (V, E)$

For any instance I, we can map it to an instance $f(I)$ of the Uranium$_b$ problem. The reduction $f : I \to f(I)$ is defined as follows. Let $k' = k + 1$. Create two new nodes A, B. Connect A to B and B to all m nodes in the second layer where each node represents a subset S_i. In the third layer, create a node for each vertex v_j and an edge between S_i and v_j if $v_j \in S_i$.

The reduction runs in polynomial time because it takes $O(\log n + \log m)$ bits to specify the vertices and at most $O(m \log n + n \log n)$ bits to specify the edges of G, giving a total running time of $O(m \log n + n \log n)$. Again, assume that we have access to the two black box algorithms.

Consider the decisional versions of the two problems. We also need to show that
There exists a set cover of size $\leq k$
$\Leftrightarrow$ There exists a subset $C \subseteq V$ (where $|C| \leq k'$) such that each vertex is either in C or has a neighbor in C.

($\Rightarrow$) Suppose there exists a set cover of size $\leq k$, a YES instance in the Set Cover problem. Then the union of all the chosen subsets is U. This corresponds to picking node B (the existence of node A ensures that B is always picked) and the k nodes in the second layer of G which corresponds to the chosen subsets, making $|C| = k' + 1$. Since every vertex in the third layer of G is covered by at least one subset node, every vertex is either in C or has a neighbor in C, a YES instance in the Uranium$_b$ problem.

($\Leftarrow$) Suppose there exists a subset $C \subseteq V$ (where $|C| \leq k'$) such that each vertex is either in C or has a neighbor in C. Then this corresponds to picking the node B to cover the edges between the first and second layer and k' nodes in the subset layer. The union of the k' nodes must contain all of the vertices or else, at least one of the vertices does not have a neighbor in C and contradict the initial assumption. Since all of the vertices are covered by the $k' - 1 = k$ subsets, this corresponds to a YES instance of the Set Cover.

(c)
The inputs of the problem are an undirected graph $G = (V, E)$, nonnegative edge weights $w(a, b)$, and a subset $Y \subseteq V$. Consider the following LP to solve Uranium$_c$

Variables:

$l(a, b)$	length of edge $e(a, b)$ where $l(a, b) \in \mathbb{R}_{\geq 0}$
$w(a, b)$	cost of edge $e(a, b)$
d_{uv}	shortest distance between vertices u, v

Objective: $\max \quad -\sum_{a,b \in E} l(a,b)w(a,b)$

Constraints:

1. For every two vertices in the set Y, their shortest distance must be at least 1.
   $\forall u, v \in Y, \qquad d_{uv} \geq 1$
2. For every two vertices u, v in V, their shortest distance is upper bounded by the distance from u to an intermediate node i and the length from i to v.
   $\forall u, v, i \in V, \qquad d_{uv} \leq d_{ui} + l(i,v)$
3. The shortest distance between a node and itself is 0.
   $\forall u \in V, \qquad d_{uu} = 0$
4. Nonnegative constraints
   $l(a,b) \geq 0, \quad w(a,b) \geq 0$

Maximizing the negative of the total cost is the same thing as minimizing the total cost. The LP is a complete description of the problem statement; constraints (2), (3), (4) guarantee the return of the shortest path just like the problem in the recitation and the additional constraint (1) ensures that the solution satisfies the problem's condition, namely the shortest distance between any two vertices in the set Y must be at least 1. The LP is polynomial because it is specified in polynomial number of constraints and variables; this can be solved using an algorithm like the ellipsoid algorithm.

2: What are Widgets, Anyways?

(a)
Variables:

$w_1, w_2, ..., w_l$	#widgets produced at each of the l factories
$D_1, D_2, ..., D_k$	total amount of each of the k materials
u_{ab}	capacity of edge $e(a, b)$
$\vec{n}_{ab} = [n_{ab,1}, ..., n_{ab,k}]$	amount of each material being sent on edge $e(a, b)$
$\vec{r}_j = [r_{1,j}, ..., r_{k,j}]$	amount of each material requested by factory f_j

Objective: $\max w_1 + w_2 + ... + w_l$
Constraints:

1. The amount of materials used to produce the widgets cannot exceed the total amount of materials available.
   $w_1 r_{1,1} + w_2 r_{1,2} + ... + w_l r_{1,l} \leq D_1$
   $w_1 r_{2,1} + w_2 r_{2,2} + ... + w_l r_{2,l} \leq D_2$
   ...
   $w_1 r_{k,1} + w_2 r_{k,2} + ... + w_l r_{k,l} \leq D_k$

2. The amount of widgets produced at each factory is limited by the supply flowing into the factory.
   Let $m_j = \min \left\lfloor \frac{\sum_t n_{tj,i}}{r_{i,j}} \right\rfloor$
   (Basically, sum up the amount of material i coming in to a factory j, find how many widgets that such an amount can potentially produce, repeat for all of the materials, and take the minimum of these). Then the constraints can be written as

   $w_j \leq m_j$, $\forall j \in \{\text{factories}\}$

3. The total outflow from each warehouse must be less than the total amount of material $\sum_j n_{vj,v} \leq D_v$ $\forall v$ warehouses

4. (Flow capacities) The total amount of material being sent on each road cannot exceed its capacity.
   $\sum_{i=1}^{k} n_{ab,i} \leq u_{ab}$, $\forall$edges $e(a, b)$

5. (Flow conservation) [Flow out of a node] - [Flow into a node] = Net supply at a node
   For all vertices $v \in V$, let j be all of the outgoing nodes from v and k be all of the incoming nodes into v.

   (a) If v is a warehouse, then for all materials $m \neq v$

   $$\sum_j n_{vj,m} - \sum_k n_{kv,m} = 0$$

(b) If v is a factory, then for all materials m

$$\sum_j n_{vj,m} \geq \sum_k n_{kv,m}$$

(c) If v is an intermediate node (non-warehouse, non-factory), then for all materials m

$$\sum_j n_{vj,m} - \sum_k n_{kv,m} = 0$$

Note that the constraints with equality can be rewritten as two inequalities

$$\sum_j n_{vj,m} - \sum_k n_{kv,m} \leq 0 \quad \text{and} \quad \sum_j n_{vj,m} - \sum_k n_{kv,m} \geq 0$$

Since there are three types of nodes in the graph (factories, warehouses, and intermediate nodes), we must consider all three. If v is a warehouse, then inflow and outflow must be the same except for the v-th material. If v is a factory, then the inflow must be at least the outflow for each material. If v is an intermediate node, then the inflow must equal the outflow.

6. Other constraints
   $w_i \geq 0 \quad u_{ab} \geq 0 \quad n_{ab,i} \geq 0$

(b)
In the above LP, add new variables $p_1, p_2, ..., p_l$ to represent the price per unit of widget produced at each factory and change the objective function to [max $w_1p_1 + ... + w_lp_l$].

(c)
Consider two special edges $e_1 = (\alpha, \beta)$ and $e_2 = (\beta, \alpha)$. We can write the new constraint as $\left|\sum_{i=1}^{k} n_{\alpha\beta,i} - \sum_{i=1}^{k} n_{\beta\alpha,i}\right| \leq \delta$ for $\delta \in \mathbb{Z}_{\geq 0}$.

Since $|x| \leq C$ can be rewritten as $x \leq C$ and $-x \leq C$, the absolute value constraint can be split into two linear constraints

1. $\sum_{i=1}^{k} n_{\alpha\beta,i} - \sum_{i=1}^{k} n_{\beta\alpha,i} \leq \delta$
2. $\sum_{i=1}^{k} n_{\beta\alpha,i} - \sum_{i=1}^{k} n_{\alpha\beta,i} \leq \delta$

3: It's in the Bag

(a)

Consider a set of n items, each of which has a size s_i and value v_i and a knapsack that has a capacity of C. Let $S(k, v)$ be the minimum knapsack size that yields at least the value v by using items from among the set $\{1, ..., k\}$. Then the recurrence for S is

$$S(k,v) = \begin{cases} 0 & \text{if } k = 0 \\ S(k-1,v) & \text{if } v_k > v \\ min\{s_k + S(k-1, v-v_k), S(k-1,v)\} & \text{else} \end{cases}$$

The algorithm can be understood as follows. If there is no item to consider, then the size is 0 (base case 1). If the value of item k exceeds the limit C of the knapsack, then we discard it and solve for $S(k-1, v)$ (base case 2). For the third case, the recurrence considers both possibilities of either using the k-th item or not, and then take the minimum of them. The optimal solution corresponds to the set that maximizes v and $S(n, v) \leq C$. Since there are only $O(nv)$ different pairs of values, we can build a table of size $n \times v$ and get a running time of $O(nv)$.

Now, we can limit the size of the table by enforcing the constraint that the size of the items being added to the knapsack must be at most C. So for every row that we fill in, as soon as we reach the first entry that exceeds C, we just stop and go on to the next row. The longest row that we need to fill in is the last row $S(n, v)$ and the cutoff point corresponds to V^*, giving us the desired running time of $O(nV^*)$.

(b)

Define the new values $v_i' := k \times \lfloor \frac{v_i}{k} \rfloor$. Let V be any feasible solution in the old instance I and V' be the corresponding feasible solution in the new instance I'. Suppose sets S, S' contain the items of the feasible solutions in I, I' respectively, then we get

1. $V' = \sum_{i \in S'} v_i' = \sum_{i \in S} k \left\lfloor \frac{v_i}{k} \right\rfloor \leq \sum_{i \in S} k \left(\frac{v_i}{k} \right) = \sum_{i \in S} v_i = V$
2. $V - V' = \sum_{i \in S} v_i - \sum_{i \in S'} v_i' = \sum_{i \in S} v_i - k \left\lfloor \frac{v_i}{k} \right\rfloor \leq \sum_{i \in S} v_i - k \left(\frac{v_i}{k} - 1 \right) = \sum_{i \in S} k \leq nk$
   $\Rightarrow V' \geq V - nk$

Therefore, $V \geq V' \geq V - nk$.
Intuitively, this makes sense because since we're scaling down by a factor of k and then rounding down, each item's value can decrease by a factor of at most k and so, the most that the value V can decrease is nk.

(c)

Let $v_i'' = \frac{v_i'}{k}$ and call this a new instance I''. Since everything is being scaled down by a factor of k, we can solve for the optimal solution of I'' and multiply it by k to get the optimal solution for the instance I'. Let V_*'' be the optimal solution for I'' and suppose that set S_*', S_*'' contains the corresponding items in the instances I', I'', respectively. Then

$$V_*'' = \sum_{i \in S_*''} v_i'' = \sum_{i \in S_*'} \frac{v_i'}{k} \leq \sum_{i=1}^{|S_*'|} \frac{v'_{max}}{k} \leq \frac{nv'_{max}}{k} \leq \frac{nv_{max}}{k}$$

By part (a) and the above analysis, we know that we can solve I' in at most $O(nV_*'') = O(\frac{n^2 v_{max}}{k})$.

(d)

By setting $k := \frac{\epsilon v_{max}}{n}$ for some $\epsilon \in (0, 1)$, the feasible solutions to I and I' are still the same since the item sizes and the knapsack size stay the same. Let V_*, V_*' be the optimal solutions of I, I' respectively. Note that $V_* \geq v_{max}$ since we assume that the most valuable item can fit in the knapsack. Then by part (b), we get

$$V_*' \geq V_* - nk = V_* - n\left(\frac{\epsilon v_{max}}{n}\right) = V_* - \epsilon v_{max} \geq V_* - \epsilon V_* = (1-\epsilon)V_*$$

This shows that the optimal value of instance I' is at least $(1-\epsilon)$ times the optimal value of I. By part (c), the running time of the second algorithm is $O\left(\frac{n^2 v_{max}}{k}\right) = O\left(\frac{n^2 v_{max}}{\epsilon v_{max}/n}\right) = O\left(\frac{n^3}{\epsilon}\right)$.