CSOR 4231 - Introduction to Algorithms HW #6

Alexander Stein (*as5281@columbia.edu*)

due 30 April 2018 at 8:00 PM

Collaborators: Matthew Carbone, Atishay Seghal

Problem 1 Statement. A large store has m customers and n products and maintains an $m \times n$ matrix A such that $A_{ij} = 1$ if customer$i$ has purchased product j; otherwise, $A_{ij} = 0$. Two customers are called *orthogonal* if they did not purchase any products in common. Your task is to help the store determine a maximum subset of orthogonal customers. Give an efficient algorithm for this problem or state its decision version and prove it is NP-complete.

Problem 1 Solution. We posit that this problem is NP-complete. Its decision version can be stated as follows:

$$\text{CUST-ORTH}(D) : \text{find a set of orthogonal customers of size} \geq k.$$

Our proof requires two steps: (1) show CUST-ORTH(D) $\in$ NP; (2) show that IS(D) $\leq_p$ CUST-ORTH(D).

Step 1: Let us have the certification algorithm CUST-ORTH-CERT(A, S, k), where A is the original matrix, the certificate S is a collection of customers s.t. $S_i \perp S_j, \forall i, j \in m \times n$, and k is the minimum number of such sets required.

Algorithm 1 certify that the given certificate solves CUST-ORTH(D)

```
procedure CUST-ORTH-CERT(A, S, k)
    if size(S) ≤ k then
        return NO
    end if
    for i = 1 to n do
        for j = 1 to m do
            if j ≠ i and S_j ⊥̸ S_i then
                return NO
            end if
        end for
    end for
    return YES
end procedure
```

It is trivial to see that this algorithm runs in $O(mn^2)$ time, which is polynomial in the size of the input. Thus, $CUST - ORTH(D) \in NP$.

Step 2: It is known that the independent-set decision problem (IS(D)) is NP-complete. Input to IS(D) is the graph $G(V, E)$, and the value k. Broadly, we'll map edges to products and vertices to customers. We define our reduction procedure, R, from IS(D) to CUST-ORTH(D) as follows:

- Let $m = |E|$ and $n = |V|$; our A will be size $n \times m$ and initially zero
- For each edge $e \in E$, give a numeric label to that edge, this will be its column-index j in A
- For each vertex $u \in V$, give a numeric label to that vertex, this will be its row-index i in A
- For each edge $e = (u, v) \in E$, $A_{ue} = 1$ and $A_{ve} = 1$, where e, u, v are indices as specified above
- the k value remains the same for both problems

Clearly this takes only polynomial time to translate the inputs. Now it remains to show that - under this translation - there is an independent-set of size $\geq k$ for IS(D) $\iff$ there is an orthogonal set of customers of size $\geq k$ for CUST-ORTH(D).

$\implies$ Suppose there is an independent-set of size k for IS(D). For all pairs of vertices selected from the generated independent set, say u and v, we have that $e = (u, v) \not\in E$. Thus in A, the entries $A_{ue} = 0$ and $A_{ve} = 0$ will be created. Since the column's of A represent edges, and one edge connects two vertices, either both $A_{ae} = 1$ and $A_{be} = 1$ or both are 0. There cannot be some $A_{ij} = 1$ and $A_{kj} = 1$ if there is no edge (i,j); And there are no edges (i,j) between any vertices i and j of the independent-set. So, under R, there will be at least k row vectors which are all mutually orthogonal.

$\impliedby$ Suppose there is a set of k orthogonal customers for CUST-ORTH(D). As described above, there can only be two 1's in the same column of A if there was originally an edge in G connecting the two vertices corresponding to these customers. For every pair a and b of customers in the orthogonal set, $A_{aj} = 1 \implies A_{bj} = 0$ (and vice. versa) for $j = 1, 2, ..., n,$. This implies that there could not have been any edges connecting the corresponding vertices of a and b in G. Thus, the set of k orthogonal customers correspond exactly to k vertices in G for which no two are connected by an edge - an independent set.

Now we may conclude that CUST-ORTH(D) $\in NPC$. $\blacksquare$

Probelm 2 Statement. Suppose you had a polynomial-time algorithm that, on an input graph, answers ***yes*** if and only if the graph has a Hamiltonian cycle. Show how, on an input graph $G = (V, E)$, you can return in polynomial time

- a Hamiltonian cycle in G, if one exists,
- ***no***, if G does not have a Hamiltonian cycle.

Problem 2 Solution. The general idea here is to examine one edge of G at a time. We look at the graph $G' = (V, E - \{e\})$ and if HAS-HAM(G') = ***yes***, then we can throw that edge away because G' still has some Hamiltonian cycle. If it returns ***no***, then we conclude that the edge we just removed must have been in every remaining Hamiltonian cycle - put it back in the graph, but mark it as visited so we do not repeat this check.

Algorithm 2 use the HAS-HAM(G) "magic" algorithm to return a Hamiltonian cycle efficiently

procedure GET-HAM($G = (V, E)$)
    if HAS-HAM(G) = ***no*** **then**
        return ***no***
    end if
    Let *crit* be a bit-vector of length $|E|$ initialized to zeros
    while $|E| > |V|$ **do**
        Let e be some edge s.t. $e \in E$ and $crit[e] = 0$
        if HAS-HAM($G' = (V, E - \{e\})$) = ***yes*** **then**
            remove e from E
        else
            $crit[e] = 1$
        end if
    end while
    return E
end procedure

First notice that we may terminate when only $|V|$ edges remain, because a simple cycle has exactly $|V|$ edges. Next, note that the *crit* indicator for an edge is only set to 1 if that edge must be in a Hamiltonian cycle. Upon termination, the E set returned will contain only those edges that are in a Hamiltonian cycle, if one exists.

If we let the running of time for HAS-HAM(G) be indicated by H(t), which we know to be polynomial, then the running time of GET-HAM(G) is $O(mH(t))$, which is also polynomial by composition.

Problem 3 Statement. There is a set of ground elements $E = e_1, e_2, ..., e_n$ and a collection of m subsets $S_1, S_2, ..., S_m$ of the ground elements (that is, $S_i \subseteq E$ for $1 \le i \le m$). The goal is to select a minimum cardinality set A of ground elements such that A contains at least one element from each subset S_i. State the decision version of this problem and prove that it is NP-complete.

Problem 3 Solution. Its decision version can be stated as follows:

$$\text{SUBSET-COVER(D) : find a subset } A \text{ as stated above of size} \le k$$

Our proof requires two steps: (1) show SUBSET-COVER(D) $\in$ NP; (2) show that VC(D) $\le_p$ CUST-ORTH(D).

Step 1: Let us have the certification algorithm SUBSET-COVER-CERT(E, S, A, k), where E is the original ground-element-set, S the original collection of subsets, the certificate is A $\subseteq$ E, and k is the maximum size of A allowed.

Algorithm 3 certify that the given certificate solves SUBSET-COVER(D)

```
1: procedure SUBSET-COVER-CERT(E, S, A, k)
2:     if size(A) ≥ k then
3:         return NO
4:     end if
5:     Let X be an array of |S| booleans, false-initialized
6:     for i = 1 to |S| do
7:         for j = 1 to |A| do
8:             if A[j] ∈ S_i then
9:                 X[i] = true
10:            end if
11:        end for
12:    end for
13:    if any element of X = false then
14:        return NO
15:    end if
16:    return YES
17: end procedure
```

Note that m is the number of sets in S, and there are at most n of them. The size of each set is obviously at most n. So the for-loop on line 4 executes n times. The inner for-loop on line 5 executes at most k times (which is at most n). The check for membership on line 6 is also at most n. All other work in this algorithm is dominated by lines 4-7, so the overall running time is $O(n^3)$, which is polynomial. Thus: SUBSET-COVER(D) $\in$ NP.

Step 2: It is known that the vertex-cover decision problem (VC(D)) is NP-complete. Input to VC(D) is the graph $G(V, E)$, and the value k. We define our simple reduction procedure, R, from VC(D) to SUBSET-COVER(D) as follows:

- the vertices of G will be the elements of E
- For each edge $e = (u, v) \in G.E$, create a new subset $S_i = \{u, v\}$ and add it to S. (The value of the index doesn't matter)
- the k value remains the same for both problems

Clearly this takes only polynomial time to translate the inputs. Now it remains to show that - under this translation - there is a vertex-cover of size $\le k$ for VC(D) $\iff$ there is a subset-cover of of size $\le k$ for

SUBSET-COVER(D).

$\Longrightarrow$ Suppose there is a vertex-cover of size k for VC(D). Under R, every set $S_i \in S$ we create will have at least one element e_v corresponding to v from that vertex-cover, because there is one set in S for every edge in $G.E$, and the vertex-cover will "touch" every edge at least once (by definition). Thus, the set A of only those elements corresponding to vertices of the vertex-cover constitutes a subset-cover of size k for SUBSET-COVER(D).

$\Longleftarrow$ Suppose there is a subset-cover A of size k for SUBSET-COVER(D). Every set $S_i \in S$ must correspond to an edge in $G.E$. Every element $e \in A$ must be correspond to a vertex in $G.V$. So, if A contains at least one element from every set, then the vertices corresponding to those elements must "touch" every edge in $G.E$ at least once. Thus, G must have had a vertex-cover of size k.

Now we may conclude that SUBSET-COVER(D) $\in NPC$. $\blacksquare$

Problem 4 Statement. A paper mill manufactures rolls of paper of standard width 3 meters. Customers watn to buy paper rolls of shorter width, and the mill has to cut such rolls from the 3m rolls. For example, one 3m roll can be cut into 2 rolls of width 93cm and one roll of width 108 cm; the remaining 6cm goes to waste. The mill receives an order of

- 97 rolls of width 135 cm
- 610 rolls of width 108 cm
- 395 rolls of width 93 cm
- 211 rolls of width 42 cm

Form a linear program to compute the smallest number of 3m rolls that have to be cut to satisfy this order, and explain how they should be cut.

Problem 4 Solution. Let us define some terms for clarity:

- Let $c_0 = 135$, $c_1 = 108$, $c_2 = 93$, and $c_3 = 42$. These are our ***cut-types***.
- Let x_i represent the total number of 3m rolls which we will cut by cut-method i (defined later).
- Let A be our coefficient matrix, where $a_{ij} = \#$ of cuts of type j for cut-method i (for a single roll)
- Let $k_0 = 97$, $k_1 = 610$, $k_2 = 395$, and $k_3 = 211$. These are the number of resized rolls ordered by each of the 4 customers.

Calculating the coefficients of A is a straightforward (albeit tedious) process. What we did to achieve this was as follows:

- List all the possible combinations of each of the cut-types. Since there are 4 types, there are $2^4 - 1 = 15$ situations to examine.
- Each such combination will be of the bounded form $a_0c_0 + a_1c_1 + a_2c_2 + a_3c_3 \leq 300$.
- We are considering unique types of combinations or ***cut-methods***, i.e. one such method says "we are cutting this 300cm roll into a_0 135-cm pieces, and a_3 42-cm pieces. As they are unique, writing out a binary-counting-decimal matrix was helpful in keeping track of what combinations were left to be calculated.
- We solved for the a_i of each cut-method by minimizing the wasted material (300 - sum), and enforcing that each coefficient take on an integer value > 0. It had to be greater than zero because otherwise it devolved into a different cut-method.
- In certain cases, there were more than one set of non-zero integer solutions. In the case that only one a-value changed from set to set, we just took the set with the largest varying value. However in the case where multiple a-values change from set to set, we added a new row to A with those constants, because such a set could potentially be used for a more optimal solution during SIMPLEX.

Here is the matrix we calculated:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 7 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 4 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$

Note that there are 15 rows, thus we will have 15 variables X_i. For example, X_{14} represents the number of rolls on which we will make 1×108 cm cut, 1×93 cm cut, and 2×42 cm cuts. Now we may finally formulate our Linear Program:

Objective:

$$\text{Minimize } \sum_{j=1}^{n} x_j$$

Subject to:

(i)

$$A^T \vec{x} \geq \vec{k}$$

(ii)

$$\vec{x} \geq \vec{0}$$

The first constraint just says ”for each cut type c_i, enforce that we create at least k_i such cuts.” Note that by the first constraint we are allowing for the possibility of creating more cuts of a particular type than required. This is necessary because requiring exact equivalence will very likely cause the LP to return ”no solution.”

Problem 5 Statement. Formulate linear or integer programs for the following optimization problems. (Full-credit will be given to LP solutions, when they are possible.)

(a) Min-cost flow: Given a flow network with capacities c_e and costs a_e on every edge e, and supplies s_i on every vertex i, find a feasible flow $f : E \to R_+$ – that is, a flow satisfying edge capacity constraints and node supplies – that minimizes total cost of the flow.

(b) The assignment problem: There are n persons and n objects that have to be matched on a one-to-one basis. There's a given set A of ordered pairs (i, j), where a pair (i, j) indicates that person i can be matched with object j. For every pair $(i, j) \in A$, there's a value a_{ij} for matching person i with object j. Our goal is to assign persons to objects so as to maximize the total value of the assignment.

(c) Uncapacitated facility location: There is a sret F of m facilities and a set D of n clients. For each facility $i \in F$ and each client $j \in D$, there is a cost c_{ij} of assigning client j to facility i. Further, there is a one-time cost f_i associated with opening and operating facility i. Find a subset F' of facilities to open that minimizes the total cost of (i) operating the facilities in F' and (ii) assigning every client j to one of the facilities in F'.

Problem 5 Solution.

(a) Min-cost flow: For this problem we can formulate a linear program. Our variables are the amount of flow f_e on each edge. Let $\vec{x}$ be an $1 \times m, m = |E|$ vector representing the flows. Similarly, $\vec{a}$ is an $1 \times m$ vector representing the edge costs, $\vec{c}$ a $1 \times m$ vector representing the capacities, and $\vec{s}$ an $1 \times n, n = |V|$ vector representing the supplies of the vertices.

Objective:

$$\text{Minimize } \vec{a}\vec{x}^T$$

Subject To:

(i)

$$\sum_{j:e_j \text{ out of } i} x_j - \sum_{j:e_j \text{ in to } i} x_j = s_i \ , \ i = 1, 2, ..., n$$

(ii)

$$\sum_{i \in V} \sum_{j:e_j \text{ out of } i} x_j - \sum_{i \in V} \sum_{j:e_j \text{ in to } i} x_j = 0 \ , \ i = 1, 2, ..., n$$

(iii)

$$\vec{x} \leq \vec{c}$$

(iv)

$$\vec{x} \geq \vec{0}$$

(b) Assignment Problem: This problem too can be formulated as a linear program, but first we must transform the inputs a little bit - into a graph! The set A we will move directly to $G.E$. The persons i and objects j will all be added to $G.V$ as individual vertices. Now it should be apparent that we have a bipartite graph, because there will be no edge between any person nodes, and no edge between any object nodes – only between person and object. The weights of each edge will equal $-a_{ij}$, because we wish to maximize "cost" instead of minimize it. Furthermore, since the problem requires an exact one-to-one correspondence, this graph problem is now a "maximum bipartite matching" problem.

Unfortunately, we're not done yet! We will translate the problem into Min-cost-flow, for which we already formulated an LP in part (a). To do so we simply (1) add the source and sink nodes s and t to $G.V$; (2) for every person node i add the zero-cost-edge (s, i) to $G.E$, and for every object node j add the zero-cost-edge (j, t) to $G.E$; (3) let the capacity of every edge c_e in the graph be 1; (4) let all supplies $s_i = 0$, $s_j = 0$ except $s_s = n$ and $s_t = -n$. When this new input is solved for min-cost-flow, we can take the person-object edges selected by that flow as our maximum value assignment.

(c) Uncapacitated facility location: For this one we are stuck with an (NP-complete) integer program formulation. We'll use $m \times n$ indicator variables:

$$x_{ij} = \begin{cases} 1 & \text{if client } j \text{ is assigned to facility } i \\ 0 & \text{otherwise} \end{cases}$$

If we define cost as "cost of opening facilities" + "cost of assigning customers" then we can define our integer program as follows:

Objective:

$$\text{Minimize: } \sum_{i=1}^{m} f_i * \max_{1 \leq j < n} x_{ij} + \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} * c_{ij}$$

Subject To:

(i)

$$\sum_{i=1}^{m} x_{ij} = 1 \text{ for } j = 1, 2, ..., n$$

(ii)

$$x_{ij} \in \{0, 1\}$$

Note that the quantity $\max_{1 \leq j < n} x_{ij} \in \{0, 1\}$ because though a facility may have multiple customers assigned to it, it is only every opened once!