6.046/18.410 Problem Set 7

Yijun Jiang
Collaborator: Hengyun Zhou, Eric Lau

November 2, 2015

1 Solving a Geology Problem Set

1.1 Part (a)

1.1.1 Box-Packing(A, M)

Proof:

We assume $m \leqslant n$, otherwise the extra $m - n$ boxes are unnecessary. Then the input size is $|x| = |(A, m)| = \Theta(n)$.

We design a verification algorithm $V_{BP}(x, y) = V_{BP}((A, m), y)$. The certificate y is a partition of A into m subsets (allowing empty subsets), such that the sum of each group does not exceed unity. The certificate size is $|y| = \Theta(n) = \Theta(|x|)$.

V_{BP} works as follows. (1) Check if y contains m subsets and if the union of these subsets equals A. (2) Sum over each group and check if all m sums are no greater than unity. If both checks are passed, return YES. Otherwise, return NO.

Check (1) costs at most $O(n^2)$ time to identify the union of all subsets with A (say neither the union nor A is ordered). Check (2) costs $O(n)$ time because $O(n)$ additions and $O(m)$ comparisons are performed, and $m \leqslant n$ is assumed. Overall, the runtime of V_{BP} is $O(n^2)$, which is a polynomial of the input size. In conclusion, Box-Packing is in NP.

1.1.2 Equal-Weight(B)

Proof:

The input size is $|x| = |B| = \Theta(n)$.

We design a verification algorithm $V_{EW}(x, y) = V_{EW}(B, y)$. The certificate y is a partition of B into 2 subsets, such that the sum of the first group equals the sum of the second. The certificate size is $|y| = \Theta(n) = \Theta(|x|)$.

V_{EW} works as follows. (1) Check if y contains 2 subsets and if the union of these subsets equals B. (2) Sum over the first group and the second group, and check if both sums are equal. If both checks are passed, return YES. Otherwise, return NO.

Check (1) costs at most $O(n^2)$ time to identify the union of both subsets with B (say neither the union nor B is ordered). Check (2) costs $O(n)$ time because $O(n)$ additions are performed. Overall, the runtime of V_{EW} is $O(n^2)$, which is a polynomial of the input size. In conclusion, Equal-Weight is in NP.

1.1.3 Desired-Weight(C, w)

Proof:

The input size is $|x| = |(C, w)| = \Theta(n + \log w)$.

We design a verification algorithm $V_{DW}(x, y) = V_{DW}((C, w), y)$. The certificate y is a subset of C, such that the sum of this subset equals w. The certificate size is $|y| = \Theta(n) = O(|x|)$.

V_{DW} works as follows. (1) Check if y is a subset of C. (2) Sum over this subset and check if the sum equals w. If both checks are passed, return YES. Otherwise, return NO.

Check (1) costs at most $O(n^2)$ time to verify a subset of C (say neither the subset nor C is ordered). Check (2) costs $O(n)$ time because $O(n)$ additions are performed. Overall, the runtime of V_{DW} is $O(n^2)$, which is a polynomial of the input size. In conclusion, DESIRED-WEIGHT is in NP.

1.2 Part (b)

1.2.1 DESIRED-WEIGHT $\leqslant_p$ EQUAL-WEIGHT

Proof:

Given an input $x = (C, w)$ for DESIRED-WEIGHT, we design a reduction function $R(x)$ that returns an input $x' = B$ for EQUAL-WEIGHT as follows. Let $w' = \sum_i c_i$. If $w' \geqslant 2w$, $R(x)$ returns $B = C \cup \{w' - 2w\}$. Otherwise, $R(x)$ returns $B = C \cup \{2w - w'\}$. $R(x)$ runs in $\Theta(n)$ time since the most expensive operation is n additions. Moreover, $|B| = \Theta(n)$ is a polynomial of $|x|$. In order to reduce DESIRED-WEIGHT to EQUAL-WEIGHT, we need to prove that DESIRED-WEIGHT returns YES if and only if EQUAL-WEIGHT returns YES.

(1) $w' \geqslant 2w$.

If $R(x) = B$ is a YES-input for EQUAL-WEIGHT, there exists a partition of $B = C \cup \{w' - 2w\}$ into two subsets, each of which sums up to $(\sum_i c_i + (w' - 2w))/2 = w' - w$. Say the newly added weight $w' - 2w$ belongs to one subset B_1. Then the rest of B_1 sums up to $(w' - w) - (w' - 2w) = w$. These rocks constitute the desired subset of C. Thus $x = (C, w)$ is a YES input for DESIRED-WEIGHT.

On the other hand, if $x = (C, w)$ is a YES-input for DESIRED-WEIGHT, there exists a subset G of C whose sum is w. Then $B_1 = G \cup \{w' - 2w\}$ and $B_2 = B - B_1$ have the same sum $w' - w$. Thus $R(x) = B$ is a YES input for EQUAL-WEIGHT.

(2) $w' < 2w$

If $R(x) = B$ is a YES-input for EQUAL-WEIGHT, there exists a partition of $B = C \cup \{2w - w'\}$ into two subsets, each of which sums up to $(\sum_i c_i + (2w - w'))/2 = w$. Say the newly added weight $2w - w'$ belongs to one subset B_1. Then the other subset B_2 is the desired subset of C that sums up to w. Thus $x = (C, w)$ is a YES input for DESIRED-WEIGHT.

On the other hand, if $x = (C, w)$ is a YES-input for DESIRED-WEIGHT, there exists a subset G of C whose sum is w. Then $B_1 = G$ and $B_2 = B - G$ have the same sum w. Thus $R(x) = B$ is a YES input for EQUAL-WEIGHT.

This completes the proof that DESIRED-WEIGHT $\leqslant_p$ EQUAL-WEIGHT. Since DESIRED-WEIGHT is NP-complete, so is EQUAL-WEIGHT. Finally, notice that the only reason for us to consider whether $w' \geqslant 2w$ is that, we are not informed if negative weights are allowed for EQUAL-WEIGHT. If they are allowed, we do not need to divide $R(x)$ into two cases.

1.2.2 EQUAL-WEIGHT $\leqslant_p$ BOX-PACKING

Proof:

Given an input $x = B$ for EQUAL-WEIGHT, we design a reduction function $R(x)$ that returns an input $x' = (A, m)$ for BOX-PACKING as follows. Let $\alpha = 2/\sum_i b_i$. Rescale $B' = [\alpha b_1, \alpha b_2, \cdots, \alpha b_n]$. Let $R(x)$ return $(A, m) = (B', 2)$. $R(x)$ runs in $\Theta(n)$ time since summation and rescaling cost this much. Moreover, $|(A, m)| = \Theta(n)$ is a polynomial of $|x|$. In order to reduce EQUAL-WEIGHT to BOX-PACKING, we need to prove that EQUAL-WEIGHT returns YES if and only if BOX-PACKING returns YES.

If $R(x) = (A, m)$ is a YES-input for BOX-PACKING, there exists a partition of $A = B'$ into $m = 2$ subsets, such that the sum of each subset does not exceed unity. Since $\sum_i a_i = \alpha \sum_i b_i = 2$, we conclude that both subsets sum up to exactly unity. This means that an equal-weight partition of B' exists. Rescaling does not change this equality. Thus $x = B$ is a YES-input for EQUAL-WEIGHT.

On the other hand, if $x = B$ is a YES-input for EQUAL-WEIGHT, then there exist a partition $B = B_1 \cup B_2$ such that B_1 and B_2 has the same sum. Correspondingly, there exist a partition $B' = B'_1 \cup B'_2$ such that B'_1 and B'_2 has the same sum. Since $A = B'$ sums up to 2, B'_1 and B'_2 each sums up to unity. So they can be packed into $m = 2$ boxes. Thus $R(x) = (A, m)$ is a YES-input for BOX-PACKING.

This completes the proof that EQUAL-WEIGHT $\leqslant_p$ BOX-PACKING. Since EQUAL-WEIGHT is NP-complete, so is BOX-PACKING.

1.3 Part (c)

Description:

If $w > nk$, return NO. Otherwise, use dynamic programming. Let $D(i, v)$ be the answer to the following True/False question: is it possible to get weight $v \in \mathbb{Z}$ from a subset of $[c_1, c_2, \cdots, c_i]$? We build up a matrix of D as follows.

Initially, set $D(0, 0) =$ True and $D(0, v) =$ False for all $1 \leqslant v \leqslant w$ (this is to make sure the edge cases are correct). Start from $i = 1$ and do the following loop: use $D(i, v) = D(i-1, v)$ OR $D(i-1, v-c_i)$ to get $D(i, v)$ for all $0 \leqslant v \leqslant w$, then increase i by 1. Iterate until $i = n$. Return $D(n, w)$ as the solution (YES for True, NO for False).

Correctness:

Runtime:

If $w > nk$, the algorithm terminates in constant time. Therefore, we focus on the case where $w \leqslant nk$. Notice that we loop over all $i \leqslant n$ and all v where $v \leqslant w \leqslant nk$. Therefore, the runtime is $\Theta(n^2 k)$. Since k is a constant, this is polynomial time of n, which is also polynomial time of the input size $\Theta(n + \log w)$. So the algorithm belongs to P.

1.4 Part (d)

Description:
Correctness:
Runtime:

1.5 Part (e)

Description:
Correctness:
Runtime:

2 Variants of Max Flow

2.1 Part (a): SWAP-FLOW

Proof:

(1) Decision version of SWAP-FLOW

Instead of finding the maximum flow across all valid capacity functions, we are given an input $g > 0$ and the YES/NO question is, does a valid capacity function c exist such that the maximum flow $|f|$ is no less than g?

(2) Construction of a reduction function $R(x)$

For more readibility, an example of this construction is shown in Fig.**??**.

Given an input $x = \phi$ for 3-SAT, we design a reduction function $R(x)$ that returns an input $x' = (G, C, g)$ for SWAP-FLOW as follows. Suppose ϕ contains m clauses $f_1, f_2, \cdots, f_m$ and n variables $x_1, x_2, \cdots, x_n$. By definition, $n \leqslant 3m$. Let $g = m + 2n$. Now construct G and C as follows.

For each variable x_i, create three vertices: p_i corresponding to the literal x_i, n_i corresponding to the literal $\neg x_i$, and a third vertex u_i. Create directed edges (u_i, p_i) and (u_i, n_i). u_i will not have any other outgoing edges. The idea is, u_i can either flow to p_i or to n_i, which stands for $x_i =$ True or False. So ideally we would set $C(u_i) = \{0, \infty\}$. But the problem forbids it: $C(u_i)$ is required to be a set of POSITIVE integers. Then we must let $C(u_i) = \{1, \infty\}$ and deal with the additional unit capacity. For each $i \leqslant n$, we connect p_i and n_i

directly to t by edges (p_i, t) and (n_i, t) to get rid of this additional unit of flow. As we see in the following, $c(p_i, t) = c(n_i, t) = 1$ by a good choice of C.

Create (s, u_i) for all $i \leqslant n$. s does not connect to any other vertices. Let $C(s) = \{\underbrace{\infty, \infty, \cdots, \infty}_{n}\}$. This guarantees that for any valid capacity function, $c(s, u_i) = \infty$. Technically speaking, $C(s)$ not a set since there are duplicate entries. But it is reasonable to assume that we are not required to use a rigorous set.

For each clause f_j, create a vertice v_j. If x_{j_α} or $\neg x_{j_\alpha}$ appears in f_j, create an edge (p_{j_α}, v_j) or (n_{j_α}, v_j), where $\alpha = 1, 2, 3$. Also, create the only outgoing edge for v_j, which is (v_j, t). Let $C(v_j) = \{1\}$. The swap here is trivial.

For each p_i or n_i, count its out-degree (denoted by d). Let $C(p_i) = \{\underbrace{1, 1, \cdots, 1}_{d_{p_i}}\}$ and $C(n_i) = \{\underbrace{1, 1, \cdots, 1}_{d_{n_i}}\}$. This guarantees that for any valid capacity function, $c(p_{j_\alpha}, v_j) = 1$ (or $c(n_{j_\alpha}, v_j) = 1$ if it is $\neg x_{j_\alpha}$ in f_j) for all $j \leqslant m$ and $\alpha = 1, 2, 3$. Moreover, $c(p_i, t) = c(n_i, t) = 1$ for all $i \leqslant n$.

This completes the construction of G. Notice that by this construction, t has in-degree $g = m + 2n$, for there are edges (v_j, t) for all $j \leqslant m$, as well as (p_i, t) and (n_i, t) for all $i \leqslant n$. All incoming edges of t have unit capacity for any valid capacity function. Therefore, the maximum flow cannot exceed g. The decision problem, in this context, is actually asking if this maximum flow can equal g.

The size and runtime of $R(x)$ are calculated here. $|V| = 3n+m+2$ and $|E| = n+2n+3m+m = 3n+4m$. So $|G| = O(|V| + |E|) = O(m + n) = O(m)$. The size of C is $O(|E|) = O(m + n) = O(m)$. The size of g is $\log(m + 2n) = O(\log m)$. Thus, the total size of $R(x)$ is $O(m)$, a polynomial of $|x|$. Each vertex and edge of G, as well as each element in $C(u)$ for each u, is created in constant time. So the runtime of $R(x)$ is linear with respect to its size. It is also $O(m)$, a polynomial of $|x|$.

(3) Equivalence of a YES-input x and a YES-input $R(x)$

As the last step, we show that $x = \phi$ is a YES-input for 3-SAT if and only if $R(x) = (G, C, g)$ is a YES-input for Swap-Flow.

First, if $R(x) = (G, C, g)$ is a YES-input for Swap-Flow, there exists a valid capacity function c such that the maximum flow f in G satisfies $|f| \geqslant g = m+2n$. As is mentioned above, it is actually $|f| = g$ that holds. All the incoming edges of t are saturated. i.e., $f(v_j, t) = c(v_j, t) = 1$ for all $j \leqslant m$, and $f(p_i, t) = c(p_i, t) = 1$, $f(n_i, t) = c(n_i, t) = 1$ for all $i \leqslant n$.

Since all capacities are integers, according to CLRS, there exists a maximum flow that is an integer flow. So WLOG suppose f is an integer flow. For each $j \leqslant m$, since $f(v_j, t) = c(v_j, t) = 1$, there must be unity flow along (p_{j_α}, v_j) or (n_{j_α}, v_j) for some $\alpha \leqslant 3$. WLOG, let $f(p_{j_1}, v_j) = 1$. By construction, x_{j_1} appears in clause f_j. We assign True to x_{j_1}, so that f_1 is satisfied. Similarly, if $f(n_{j_1}, v_j) = 1$, $\neg x_{j_1}$ appears in clause f_j and we assign False to x_{j_1}.

By doing this assignment to all $j \leqslant m$, every clause is satisfied and so is ϕ. We claim that this assignment is self-consistent. This is because, say $f(p_{j_1}, v_j) = 1$ for some j (which assigns True to x_{j_1}), we must have $f(u_{j_1}, p_{j_1}) \geqslant f(p_{j_1}, v_j) + f(p_{j_1}, t) = 2$. Therefore, $c(u_{j_1}, p_{j_1}) \geqslant 2$. Since $C(u_{j_1}) = \{1, \infty\}$, it must be that $c(u_{j_1}, p_{j_1}) = \infty$ and $c(u_{j_1}, n_{j_1}) = 1$. n_{j_1} cannot flow to any $f_{j'}$ because $f(n_{j_1}, t) = 1$ has taken up this unit of capacity. So x_{j_1} will never be assigned False. Similarly, if x_{j_1} is assigned False, it will never be assigned True again.

After this procedure, there may be some x_i left unassigned. They can be assigned either to True of to False, for their values do not matter. The overall assignment satisfies ϕ. Therefore, $x = \phi$ is a YES-input for 3-SAT.

Next, we prove the opposite direction. If $x = \phi$ is a YES-input for 3-SAT, there exists a assignment of True or False to x_i for all i such that ϕ is satisfied. According to this assignment, we show that there is a valid capacity function c for Swap-Flow that makes $|f| = g$. This capacity function works as follows. For each i, if x_i is assigned True, then $c(u_i, p_i) = \infty$ and $c(u_i, n_i) = 1$. Otherwise, $c(u_i, p_i) = 1$ and $c(u_i, n_i) = \infty$. $c(s, u_i) = \infty$ for all i, and the capacities for the rest of the edges are set to unity (the swap is nontrivial only at u_i).

unfinished

2.2 Part (b): All-Or-None-Flow

Proof:
(1) Decision version of All-Or-None-Flow

Instead of finding the maximum flow under the restriction that $f(e) = 0$ or $f(e) = c(e)$ for all edges e, we are given an input $g > 0$ and the YES/NO question is, does a flow f under this restriction exist such that $|f| \geqslant g$?

(2) Proof of All-Or-None-Flow $\in$ NP

The input is $x = (G, c(e), g)$, where $G = (V, E)$ is the flow network and $c(e)$ is the capacity function that assigns each edge a positive capacity. The input size is $|x| = \Theta(|V| + |E|)$.

We design a verification algorithm $V_{AONF}(x, y) = V_{ANOF}((G, c(e), g), y)$. The certificate y is a flow f over G such that $f(e) = 0$ or $f(e) = c(e)$ for all $e \in E$, and $|f| \geqslant g$ holds. The certificate size is $|y| = \Theta(|E|)$ because the flow is determined by its values on all edges. It is a polynomial of $|x|$.

V_{ANOF} works as follows. (1) Check that $f(e)$ is either 0 or $c(e)$ for all $e \in E$. (2) Check that $|f| \geqslant g$. If both checks are passed, return YES. Otherwise, return NO.

Check (1) costs $\Theta(|E|)$ time because it loops over E. Check (2) costs $O(|V|)$ time if we use $|f| = \sum_u f(s, u)$. Overall, the runtime of V_{ANOF} is $O(|V| + |E|)$, which is a polynomial of the input size. In conclusion, All-Or-None-Flow is in NP.

(3) Construction of a reduction function $R(x)$

Given an input $x = B = \{b_1, b_2, \cdots, b_n\}$ for Equal-Weight, we design a reduction function $R(x)$ that returns an input $x' = (G, c(e), g)$ for All-Or-None-FLow as follows. Create a source s, a sink t and an intermediate vertex m. Create n edges from s to m. Label them as e^i_{sm}, where $i = 1, 2, \cdots, n$, and let $c(e^i_{sm}) = 2b_i$. Also create n edges from m to t. Label them as e^i_{mt}, where $i = 1, 2, \cdots, n$, and let $c(e^i_{mt}) = b_i$. Let $g = \sum_i b_i$. $|R(x)| = \Theta(|V| + |E|) = \Theta(n)$, which is a polynomial of $|x| = \Theta(n)$. The runtime of $R(x)$ is also $\Theta(|V| + |E|)$, which is a polynomial of $|x|$. In order to reduce Equal-Weight to All-Or-None-Flow, we need to prove that Equal-Weight returns YES if and only if All-Or-None-Flow returns YES.

(4) Proof of Equal-Weight $\leqslant_p$ All-Or-None-Flow

If $R(x) = (G, c(e), g)$ is a YES-input of All-Or-None-Flow, there exists a flow f such that $|f| \geqslant g$. By construction, $\sum_i c(e^i_{mt}) = \sum_i b_i = g$, so $|f| \leqslant g$. Therefore, it must be that $|f| = g$.

This means that $\sum_i f(e^i_{sm}) = g$. Notice that $\sum_i c(e^i_{sm}) = \sum_i 2b_i = 2g$. Because f is restricted such that $f(e)$ is either 0 or $c(e)$, we conclude that some edges between s and m are fully utilized and the others are empty. Let $B_1 = \{b_j | e^j_{sm}$ is fully utilized$\}$ and $B_2 = B - B_1$. $\sum_{b_j \in B_1} 2b_j = \sum_{b_j \in B_1} c(e^j_{sm}) = |f| = g$. Therefore, B_1 sums up to $g/2$, and so does B_2. This means that B can be partitioned into two equi-sum subsets. So $x = B$ is a YES-input for Equal-Weight.

On the other hand, if $x = B$ is a YES-input of Equal-Weight, there exists a partition of B into two subsets, B_1 and B_2, such that the sum over B_1 equals the sum over B_2.

For each j such that $b_j \in B_1$, we push $c(e^j_{sm}) = 2b_j$ flow through the edge e^j_{sm}. All other edges between s and m are not used. This produces an incoming flow of $\sum_{b_j \in B_1} 2b_j = \sum_i b_i = g$For all edges e^i_{mt}, we push $c(e^i_{mt}) = b_i$ flow through them