CSC373— Algorithm Design, Analysis, and Complexity — Spring 2018

Solutions for Tutorial Exercise 7: Circulations and Polynomial Reductions

1. **Existence of a Circulation.** On slide 7 of the lecture notes on circulations and flows we stated:

    Characterization. Given a circulation problem (V, E, c, d) there does **not** exist a circulation iff there exists a partition (A, B) of the vertices V (i.e., with $B = V \backslash A$) such that $\sum_{v \in B} d(v) > \text{cap}(A, B)$.

    The capacity of this partition (A, B) is defined to be $\text{cap}(A, B) \equiv \sum_{e \in A2B} c(e)$, where $A2B \equiv \{e \in E \mid e = (u, v), u \in A, v \in B\}$. Note this is similar to the capacity of s-t cuts except, for circulations, A and B are not restricted to contain s and t, respectively. Also, note that $\text{cap}(A, B)$ is not generally equal to $\text{cap}(B, A)$.)

    Soln 1. Let $G' = (V', E')$ be the corresponding s-t flow problem described on slide 5 of lecture notes linked above. Then $V' = V \cup \{s, t\}$ and E' is E, plus all edges (s, u) for vertices $u \in V$ with $d(u) < 0$, and all edges (w, t) for vertices $w \in V$ with $d(w) > 0$. The capacity of these added edges s are $c((s, u)) = -d(u) > 0$. Similarly, each (w, t) edge has capacity $c((w, t)) = d(w) > 0$. Also from the lecture notes, we know that the circulation (V, E, c, d) does not exist iff the max flow of this G' is less than D, where $D = \sum_{\{w \in V \mid d(w) > 0\}} -d(w) = \sum_{\{u \in V \mid d(u) < 0\}} -d(u)$.

    Show $\Rightarrow$ direction. Suppose there does not exist a circulation. Then, from the lecture notes, we know that the s-t flow problem G' must have a max flow with value $\nu(f) < D$ (so some of the edges leaving s cannot be saturated).

    Let f be a max flow for G', and consider a minimum capacity s-t cut (A', B'). By the definition of s-t cuts, $s \in A'$ and $t \in B'$. And by the max-flow min-cut theorem, we have $\nu(f) = \text{cap}(A', B')$, so we have $D > \text{cap}(A', B')$.

    Define $A = A' \backslash \{s\}$ and $B = B' \backslash \{t\}$. Then it follows that (A, B) is a partition of the vertices V of the circulation problem. We are left with showing that this partition has the desired property. First note, by the construction of A' and B' we have

$$\begin{aligned} D &> \text{cap}(A', B'), \\ &= \left[\sum_{\{e=(s,w) \in E' \mid w \in B\}} c(e)\right] + \left[\sum_{\{e=(u,v) \in E \mid u \in A,\, v \in B\}} c(e)\right] + \left[\sum_{\{e=(u,t) \in E' \mid u \in A\}} c(e)\right], \\ &= \left[\sum_{\{w \in B \mid d(w) < 0\}} -d(w)\right] + \text{cap}(A, B) + \left[\sum_{\{u \in A \mid d(u) > 0\}} d(u)\right]. \end{aligned} \tag{1}$$

    But expanding the demand D gives the following,

$$\begin{aligned} D &= \sum_{\{u \in V \mid d(u) > 0\}} d(u), \\ &= \sum_{\{u \in A \mid d(u) > 0\}} d(u) + \sum_{\{w \in B \mid d(w) > 0\}} d(w), \quad \text{since } (A, B) \text{ is a partition of } V. \end{aligned} \tag{2}$$

    Combining (2) with (1) and cancelling common terms gives

$$\sum_{\{w \in B \mid d(w) > 0\}} d(w) > \left[\sum_{\{w \in B \mid d(w) < 0\}} -d(w)\right] + \text{cap}(A, B), \tag{3}$$

    so the desired result, namely $\sum_{\{w \in B\}} d(w) > \text{cap}(A, B)$, now follows.

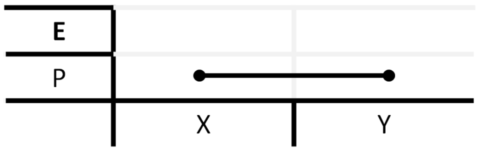
    Show $\Leftarrow$ direction. Suppose (A, B) is a partition of the vertices V of the circulation problem such that $\sum_{\{w \in B\}} d(w) > \text{cap}(A, B)$. Define $A' = A \cup \{s\}$ and $B' = B \cup \{t\}$, and the s-t flow problem G' as above. Then (A', B') is an s-t cut for G'.

Moreover, working backwards through inequalities (3), (2), and (1), respectively, we find that all these inequalities must all hold. Therefore, $\text{cap}(A', B') < D$.

By the weak duality property, the value of the maximum flow $\nu(f)$ must satisfy $\nu(f) \leq \text{cap}(A', B')$ and therefore $\nu(f) < D$. Since the the sum of capacities of edges leaving s is $\text{cap}(\{s\}, V'\{s\}) = D$, the max flow of G' does not have a flow which saturates all the edges leaving s. Therefore, by the result in the lecture notes (slide 5), a circulation for (V, E, c, d) does not exist. ■

2. **Picturing What Poly-Reductions Tell Us.** Suppose $X(s)$ and $Y(s)$ are two decision problems. We would like to know if these problems have a polynomial time solution or are they inherently exponential? That is, can $X(s)$ be solved with a deterministic algorithm that runs in $O(|s|^q)$ time, for some constant q? That is, is $X(s) \in P$? Or is it the case that, for every constant $q > 0$, the worst case runtime of any deterministic algorithm for solving $X(s)$ is necessarily $\Omega(|s|^q)$? We refer to this latter class as E (for exponential).

   (a) What do we learn about the possible classifications of X and Y if we show $X \leq_p Y$? For example, it could be the case that both X and Y have polytime solutions. We can depict this pair of possibilities as the line segment in the figure below.

   Draw all other possible pairs in the figure above. Are any pairs ruled out? Explain.

   Soln 2a. There are three edges, in the form of a Z. That is, the only pair that is ruled out is $X(s)$ in E and $Y(s)$ in P. The reason is that, given a polynomial reduction showing $X \leq_p Y$ and $Y \in P$, then this reduction actually provides a polynomial time algorithm for solving X.

   (b) In the lectures we showed

   $$\text{3-SAT} \leq_p \text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$$

   If any of these turn out to have a polytime solution, which others must also have polytime solutions? Similarly, if any of these are shown to be in E, as defined above, which others must be in E? Briefly explain.

   Soln 2b. Suppose you plot the possiblities as in the figure above, with one column for each of the decision problems, and with the problems given in the order listed above. In each column we place a vertex either row P or E to denote the class of the runtime necessary for the solution, and connect vertices in neighbouring columns with an edge. Each feasible placement of vertices gives a path with three edges. From the argument in part (1a) we know this path can increase (i.e., go from row P to E) only as you move to the right. Note the edge between INDEPENDENT-SET and VERTEX-COVER must always be horizontal, due to their $\equiv_p$ relationship.

   For example if, on the far right, the vertex for SET-COVER is in row P then all the other vertices must be in P. Alternatively, if the vertex for 3-SAT is in E (which, by the way, is the consensus view) then all the vertices must be E.

3. **Set Packing with Sets.** The set packing decision problem is defined in a similar way to the definition provided below. Here we have only replaced the notion of a collection/family/list of subsets F with a simple set of subsets F:

   SetPack: Given a universe set U, a set of subsets $F = \{S_j \mid S_j \subseteq U,\ 1 \leq j \leq m\}$, and an integer k, does there exist $C \subseteq F$ with $|C| \geq k$ such that no two distinct elements $S_i, S_j \in C$ intersect (i.e., for all S_i, S_j in C with $S_i \neq S_j$ we have $S_i \cap S_j = \emptyset$)?

   (a) Denote the independent set decision problem by **IndepSet**. Show **IndepSet** $\leq_p$ **SetPack**.

   (b) Show **SetPack** $\leq_p$ **IndepSet**.

(c) Define **searchSetPack** to be the search problem for set packing. That is, given U and F as in the Set-Packing decision problem, **find a subset** $C \subseteq F$ such that $|C|$ is the maximum possible and no two distinct elements in C intersect.

Prove that **searchSetPack** $\leq_p$ **SetPack**.

Hint 1: You need to first find k^*, the maximum possible size $|C|$. Then find the elements of C.

Hint 2: For finding the elements of C it is useful to write a loop invariant stating that the current solution "is promising".

Solution for Q3:

3a **Soln.** We will show that the set packing (with sets) problem is a generalization of the independent set problem.

The input provided to **IndepSet** is an undirected graph $G = (V, E)$ and an integer k. We will measure the size of this input to be $|s| = |V| + |E|$ (you can define anything reasonable here, but it is critical to specify $|s|$ in order to make the argument that various algorithms are poly-time in $|s|$).

The independent set problem concerns the existence of a subset of the vertices, say $C \subset V$, such that $|C| \geq k$ and no two vertices in C share an edge. Note that we can express this latter constraint in terms of the sets of edges ending at each vertex, i.e.,

$$S_v = \{e = (u, v) \mid \text{for some } u \in V \text{ and } e \in E\}. \tag{4}$$

This observation forms the basis of the poly-time reduction.

One difficulty we might anticipate in using these S_v's as the individual sets in a packing is that there can be multiple empty sets in this construction. Indeed $S_v = \emptyset$ for any vertex v that is not an endpoint for any edges in E. Moreover, multiple such vertices can participate in an independent set. For example, if we define F to be the **set** of all such S_v then, in the case of a graph with no edges, we have $F = \{\emptyset\}$ and so $|F| = 1$. Meanwhile, the maximum independent set for this case consists of all V.

Another possible difficulty is that for cases in which $u, v \in V$, with u and v endpoints of only the one edge $e = (u, v)$, then $S_u = S_v = \{e\}$. Here again, the sets S_v would not be in one to one correspondence with the set of vertices.

The issue here is simply that instead of considering a list of subsets (or a "collection" of subsets), where the subscript v on S_v matters, here we are treating S_v simply as an element of the set F.

It is therefore convenient to include the specific endpoint vertex in each of the subsets to be used for packing. That is, we define

$$F_v = \{v\} \cup S_v, \text{ and } F = \{F_v \mid v \in V\}. \tag{5}$$

Here there is exactly one element of F for every v, so $|F| = |V|$. And we define the universe set to be $U = V \cup E$.

Claim: For the above construction of U and F we have **IndepSet**(G, k) iff **SetPack**(U, F, k). Note we use the same k for both problems.

Assuming this Claim for the moment, note that, given the input (G, k) of the **IndepSet** problem, we can construct U and F in at most $O(|V||E|)$ time. By our definition of $|s|$ above, this is bounded by $O(|s|^2)$ time. In addition, according to the claim, we require only one call to **SetPack**(U, F, k) to determine the answer for **IndepSet**(G, k). Therefore, by the definition of a poly-time reduction, we have **IndepSet**(G, k) $\leq_p$ **SetPack**(U, F, k), as desired.

Proof of Claim:

$\Longrightarrow$ Suppose **IndepSet**(G, k) is true. Let $C \subseteq V$ be an independent set of size $|C| \geq k$. Define $F_C = \{\{v\} \cup S_v \mid v \in C\}$. Then by (4) and (5) we have $|F_C| = |C| \geq k$. Let F_a and F_b be any two distinct elements in F_C, that is, $a \neq b$. Since $a, b \in C$ and C is an independent set, there are no edges in E which have both a and b as endpoints. Therefore $S_a \cap S_b = \emptyset$. It now follows from (5) that $F_a \cap F_b = \emptyset$. Therefore F_C is a set packing, and we've already shown $|F_C| = |C| \geq k$, so **SetPack**(U, F, k) is true.

$\Longleftarrow$ Suppose **SetPack**(U, F, k) is true. Then there exists a set-packing of size at least k. Since the elements of F are in one to one correspondence with the elements of V we can, without loss of generality, write such a set packing as $F_C = \{\{v\} \cup S_v \mid v \in C\}$ where $C \subseteq V$ and $|C| = |F_C| \geq k$.

Given any two distinct vertices a and b in C, i.e., with $a \neq b$, we know F_a and F_b are distinct elements in F and, by the definition of the set packing problem, we know $F_a \cap F_b = \emptyset$. Now, by the definition of F_v, this implies $S_a \cap S_b = \emptyset$. That is, a and b cannot be endpoints of the same edge in E.

Since this is true for any such $a \neq b$ in C, we conclude that C is an independent set. Recall $|C| = |F_C| \geq k$, so C is an independent set of G of size at least k. Therefore **IndepSet**(G, k) is true.

3b **Soln.** Left to the reader.

3c **Soln.** We wish to have **searchSetPack**(U, F) return a maximum size subset $M \subseteq F$ such that, for each distinct pair of elements M_i and M_j in M, we have $M_i \cap M_j = \emptyset$.

Define the size of the input to be $|s| = |U| + |F|$ (again, make any reasonable choice, but make that choice clear). By using binary search on k and calling **SetPack**(U, F, k) at most $O(\log(|F|)) \subset O(\log(|s|))$ times, determine the maximum size $k^* = |M|$ for a set-packing M (details omitted). Given this k^* we then execute the following:

```
Suppose $F = \{F_1, F_2, \ldots, F_m\}$, in any order, where $m = |F|$.
Initialize $M \leftarrow \emptyset$ and $k \leftarrow k^*$
for j = 1..m:
    # Loop Invariant: LI(j): There exists a maximum set-packing $T \subset F$, with $|T| = k^*$
    # such that, $F_i \in T$ iff $F_i \in M$ for $1 \leq i < j$ (i.e., $M$ is promising).
    # Moreover $|M| = k^* - k$.
    if $k == 0$:
        break # No more elements to find.
    $C \leftarrow (\bigcup_{M_i \in M} M_i)$ # Elements covered so far.
    if $F_j \cap C = \emptyset$:
        # Determine whether there exists a solution $T' = M \cup F_j \cup T'_{j+1}$ of size $k^*$,
        # with $T'_{j+1} \subseteq \{F_{j+1}, \ldots, F_m\}$ and $|T'_{j+1}| = k^* - |M| - 1 = k - 1$.
        $C' \leftarrow C \cup F_j$
        $F' = \{F_i \mid j < i \leq m$ and $F_i \cap C' = \emptyset\}$.
        if SetPack$(U, F', k - 1)$:
            $k \leftarrow k - 1$
            $M \leftarrow M \cup F_j$
return M
```

Here we only sketch the remainder of the proof. We use induction to prove the loop invariant $LI(j)$ for $j = 0, 1, \ldots, J$, where J is the index j for which the break statement is executed or, if the break is not executed, then $J = m + 1$. In the latter case, we define $LI(m + 1)$ as the loop invariant **after** the m^{th} execution of the loop.

The proof of the loop invariant follows an "is promising" style proof (as discussed in the lectures on greedy algorithms). In the situations where you need to switch from the optimal solutions T in the loop invariant $LI(j)$ to a new optimal solution (i.e., use an exchange argument, or"switch horses" in mid-proof) use the T' described in the algorithm above.