Lab04-Solution

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

$*$ If there is any problem, please contact TA Yiming Liu.

1. Give a directed graph $G = (V, E)$ whose edges have integer weights. Let $w(e)$ be the weight of edge $e \in E$. We are also given a constraint $f(u) \geq 0$ on the out-degree of each node $u \in V$. Our goal is to find a subset of edges with maximal weight, whose out-degree at any node is no greater than the constraint.

   (a) Please define independent sets and prove that they form a matroid.

   (b) Write an optimal greedy algorithm based on Greedy-MAX in the form of *pseudo code*.

   (c) Analyze the time complexity of your algorithm.

   Solution. The solutions are as follows:

   (a) Define the independent sets as all subsets of edges whose out-degree does not exceed $f(u)$ at each node u. For an independent set F, define $d_F(u)$ the number of out-edges at u that belong to F. By definition of independent sets, we have $d_F(u) \leqslant f(u)$ for each node u.

   The hereditary property is obvious because removing some edge whose source is a node u will decrease $d_F(u)$ and further relax the constraint on $f(u)$. As for the exchange property, consider two independent sets of edges A and B, with $|A| < |B|$. Because there are fewer edges in A there must exist a node u that is the source of fewer edges in A than in B, i.e., such that $d_A(u) < d_B(u)$. Let e be an edge in $B \backslash A$ whose source is u. We can safely add e to A without violating the degree constraint $f(u)$ because $d_A(u) + 1 \leqslant d_B(u) \leqslant f(u)$. Hence, $A \cup \{e\}$ is independent.

   (b)

```
Algorithm 1: GreedyMax-01
Input: $G = (V, E)$ and the constraints $f(u)$ of every node $u$
Output: A subset $S$ of edges with maximal weight, whose out-degree at
        any node is no greater than the constraint.
1 $S = \emptyset$;
2 Sort all edges into ordering $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_n)$;
3 for $i = 1$ to $n$ do
4     if $S \cup \{e_i\} \in \mathcal{I}$ then
5         $S = S \cup \{e_i\}$;
6 return $S$;
```

   (c) Checking whether an edge can be added can be done in constant time, so the complexity of the greedy algorithm is dominated by the time needed to sort the edges by nonincreasing weights, which requires $O(|E| \log |E|)$ steps.

   $\square$

2. Let X, Y, Z be three sets. We say two triples (x_1, y_1, z_1) and (x_2, y_2, z_2) in $X \times Y \times Z$ are *disjoint* if $x_1 \neq x_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$. Consider the following problem:

   Definition 1 (MAX-3DM). *Given three disjoint sets X, Y, Z and a nonnegative weight function $c(\cdot)$ on all triples in $X \times Y \times Z$, **Maximum 3-Dimensional Matching** (MAX-3DM) is to find a collection $\mathcal{F}$ of disjoint triples with maximum total weight.*

(a) Let $D = X \times Y \times Z$. Define independent sets for MAX-3DM.

(b) Write a greedy algorithm based on Greedy-MAX in the form of *pseudo code*.

(c) Give a counterexample to show that your Greedy-MAX algorithm in Q. 2b is not optimal.

(d) Show that: $\max_{F \subseteq D} \frac{v(F)}{u(F)} \leq 3$. (Hint: you may need Theorem 1 for this subquestion.)

Theorem 1. *Suppose an independent system $(E, \mathcal{I})$ is the intersection of k matroids $(E, \mathcal{I}_i)$, $1 \leq i \leq k$; that is, $\mathcal{I} = \bigcap_{i=1}^{k} \mathcal{I}_i$. Then $\max_{F \subseteq E} \frac{v(F)}{u(F)} \leq k$, where $v(F)$ is the maximum size of independent subset in F and $u(F)$ is the minimum size of maximal independent subset in F.*

Solution. The solutions are as follows:

(a) A subset $I \subseteq D$ is independent if and only if each two triples in I are disjoint.

(b)

Algorithm 2: GreedyMax-02

```
Input: $X, Y, Z$ and the weight function $c(\cdot)$ on all triples in $X \times Y \times Z$
Output: A collection $\mathcal{F}$ of disjoint triples with maximum total weight
1 Sort all triples in $D$ into ordering $c(d_1) \geq c(d_2) \geq ... \geq c(d_m)$.
2 $\mathcal{F} \leftarrow \varnothing$;
3 for $i = 1$ to $m$ do
4     if $\mathcal{F} \cup \{d_i\} \in$ C then
5         $\mathcal{F} \leftarrow \mathcal{F} \cup \{d_i\}$
6 return $\mathcal{F}$;
```

(c) Suppose $X = \{a, b\}, Y = \{c, d\}, Z = \{e, f\}$, and we define $c(\cdot)$ as follows:

$$c((a,c,e)) = 2, \quad c((a,c,f)) = 3, \quad c((a,d,e)) = 1, \quad c((a,d,f)) = 1$$
$$c((b,c,e)) = 1, \quad c((b,c,f)) = 1, \quad c((b,d,e)) = 0, \quad c((b,d,f)) = 2$$

The answer is $\mathcal{F} = \{(a,c,f), (b,d,e)\}$ by Algorithm 2 and the total weight of $\mathcal{F}$ is $3 + 0 = 3$. Howerver, the optimal solution is $\mathcal{F}^* = \{(a,c,e), (b,d,f)\}$ and the total weight of $\mathcal{F}^*$ is $2 + 2 = 4$.

(d) For given sets X, Y, and Z, let $E = X \times Y \times Z$. Also, let $\mathcal{I}_X$ $(\mathcal{I}_Y, \mathcal{I}_Z)$ be the family of subsets A of E such that no two triples in any subset share an element in $X (Y, Z,$ respectively). Then $(E, \mathcal{I}_X), (E, \mathcal{I}_Y)$, and $(E, \mathcal{I}_Z)$ are three matroids and MAX- 3 DM is just the problem of finding the maximum-weight intersection of these three matroids. By Theorem 1 we see that $\max_{F \subseteq D} \frac{v(F)}{u(F)} \leq 3$.

□

3. **Crowdsourcing** is the process of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, especially an online community. Suppose you want to form a team to complete a crowdsourcing task, and there are n individuals to choose from. Each person p_i can contribute v_i $(v_i > 0)$ to the team, but he/she can only work with up to c_i other people. Now it is up to you to choose a certain group of people and maximize their total contributions $(\sum_i v_i)$.

   (a) Given OPT(i, b, c) = maximum contributions when choosing from $\{p_1, p_2, \cdots, p_i\}$ with b persons from $\{p_{i+1}, p_{i+2}, \cdots, p_n\}$ already on board and at most c seats left before any of the existing team members gets uncomfortable. Describe the optimal substructure as we did in class and write a recurrence for OPT(i, b, c).

(b) Design an algorithm to form your team using dynamic programming, in the form of *pseudo code*.

(c) Analyze the time and space complexities of your design.

Solution. (a) There are three non-trivial cases to consider.

1. OPT cannot choose p_i because $c_i < b$. Then OPT chooses best of $\{p_1, p_2, \cdots, p_{i-1}\}$ with b persons already on board and at most c seats left.
2. OPT does not choose p_i but $c_i \geq b$. Then OPT chooses best of $\{p_1, p_2, \cdots, p_{i-1}\}$ with b persons already on board and at most c seats left.
3. OPT chooses p_i and thus $c_i \geq b$. Then OPT chooses best of $\{p_1, p_2, \cdots, p_{i-1}\}$ with $b+1$ persons already on board and at most $\min\{c-1, c_i - b\}$ seats left.

Therefore, we can write a recurrence for $\mathrm{OPT}(i, b, c)$ as follows.

$$\mathrm{OPT}(i,b,c) = \begin{cases} 0 & \text{if } i = 0 \text{ or } c = 0 \\ \mathrm{OPT}(i-1,b,c) & \text{if } c_i < b \\ \max\{\mathrm{OPT}(i-1,b,c), v_i + \mathrm{OPT}(i-1,b+1,c-1)\} & \text{if } c-1 \leq c_i - b \\ \max\{\mathrm{OPT}(i-1,b,c), v_i + \mathrm{OPT}(i-1,b+1,c_i-b)\} & \text{otherwise} \end{cases}$$

(b) The pseudo code should be straightforward.

Algorithm 3: Team Formation

```
Input: Each person's contribution $V[1, \cdots, n]$ and tolerance $C[1, \cdots, n]$
Output: The maximum total contributions of the team

for $i \leftarrow 0$ to $n$ do
    for $b \leftarrow 0$ to $n - i$ do
        for $c \leftarrow 0$ to $n - b$ do
            if $i = 0$ or $c = 0$ then $OPT[i,b,c] \leftarrow 0$;
            else if $C[i] < b$ then $OPT[i,b,c] \leftarrow OPT[i-1,b,c]$;
            else
                $OPT[i,b,c] \leftarrow$
                $\max(OPT[i-1,b,c], V[i] + OPT[i-1,b+1,\min(c-1, C[i]-b)])$;
return $OPT[n,0,n]$;
```

(c) Both of the time and space complexities of my design is $O(n^3)$. Also, you can have your own design.

□

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.