Lab04-Matroid

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

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

* Name:Haotian Xue Student ID:518021910506 Email: xavihart@sjtu.edu.cn

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 of Problem 1

(a) **Definition of independent sets:** Define the edge set $A \subset E$ and the vertex set of A is V_A, also $V_A \subset V$, if for any $u \in V_A$ we have the out degree of u in edge set A is no greater than $f(u)$, we can say A is an independent set. We define the set of independent sets C. We want to prove the $\mathcal{M} = (E, C)$ is a matroid. We define the vertex set of an edge set X is V_X, and the out degree of vertex u int V_X is $out(u, X)$.

Proof. It is easy to prove the **hereditary** of $\mathcal{M} = (E, C)$, since for any $A \subset B, B \in C$, for any vertex $u \in V_B$, $u \in V_A$, we can get: $f(u) \geq out(u, B) \geq out(u, A)$. So A is also an independent set, $A \subset C$.

To prove the **exchange property**, we will use an obvious lemma: for edge set X, we have:

$$\sum_{u \in V_X} out(u, X) = |X|$$

Then, for $A, B \subset C$, $|A| < |B|$, we have: $\sum\limits_{u \in V_A} out(u, A) < \sum\limits_{u \in V_B} out(u, B)$, there must exists one vertex u_0, such that $out(u_0, A) < out(u_0, B) \leq f(u_0)$, which means that the number of edges start from u_0 in A is less than that in B, so we can find an edge $x \in B - A$, define $A^{'} = A \cup \{x\}$, $out(u_0, A^{'}) = out(u_0, A) + 1 \leq f(u_0)$. So $A^{'} \subset C$.

Considering the two properties above, we have $\mathcal{M} = (E, C)$ is a matroid. □

(b)The pseudo code of MAX-Greedy Algorithm.

Input: Directed graph $G = (V, E)$, constraint list
$F = \{f(u_1), f(u_2), ..., f(u_{|V|})\}$, weight list
$W = \{w(e_1), w(e_2), ..., w(e_{|E|})\}$

Output: Edge set E_{max} which has the maximum weight.

$E_{max} = \{\}$, $OutDegreeMap = \{v_1 : 0, v_2 : 0, ..., v_{|V|} : 0\}$
Sort the edges by weight in an decreasing order.
for $i = 1$ *to* $|E|$ **do**
    $u =$ start vertex of e_i
    if $OutDegree(u) + 1 \leq f(u)$ **then**
        $E_{max} = E_{max} \cup \{e_i\}$
        OutDegree(u) = OutDegree(u) + 1
return E_{max}

(c) The sort part in the algorithm has a time complexity of $O(|E| \log |E|)$. The loop part is O(|E|) since in each loop, we use a map to query and change value of out degrees. So the final time complexity is $O(|E| \log |E|)$.

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: $\max\limits_{F \subseteq D} \frac{v(F)}{u(F)} \leq 3$. (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 $\max\limits_{F \subseteq E} \frac{v(F)}{u(F)} \leq k$, 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 of Problem 2

(a)

We consider the set as a 3-D matches between X, Y, Z. The set A is an independent set if and only if every two matches in A are disjoint with each other.

(b)

(x_1, y_1, z_1)	100
(x_1, y_1, z_2)	10
(x_1, y_2, z_1)	10
(x_1, y_2, z_2)	60
(x_2, y_1, z_1)	55
(x_2, y_1, z_2)	40
(x_2, y_2, z_1)	30
(x_2, y_2, z_2)	10

Table 1: Counterexample

Input: Three sets: $X = \{x_1, ..., x_{n_X}\}, Y = \{y_1, y_2, ..., y_{n_Y}\}, Z = \{z_1, z_2, ..., z_{n_Z}\}$ and the weight matrix $W[n_X][n_Y][n_Z]$

Output: The indepent set with the max weight: C

$C = \{\}$

Construct the list of triple $Triples = \{(x_i, y_j, z_k)\}$ for $i = 1, 2, ..., n_X$, $j = 1, 2, ..., n_Y$, $k = 1, 2, ..., n_Z$.

Sort $Triples$ by weights of triples in an decreasing order.

for i = *1 to* $n_X n_Y n_Z$ **do**

  if $A[i]$ *is disjoint with any triples in* C **then**

    $C = C \cup \{Triple[i]\}$

return C

(c)

Counterexample : $X = \{x_1, x_2\}, Y = \{y_1, y_2\}, Z = \{z_1, z_2\}$, the weights of the triples is shown in Table 1.

Using the Greedy-MAX algorithm, we first choose (x_1, y_1, z_1) which has the maximal weight, then we can only choose (x_2, y_2, z_2). So the independent set accquired is $\{(x_1, y_1, z_1), (x_2, y_2, z_1)\}$ which has a weight of 110, however the independent set $\{(x_1, y_2, z_2), (x_2, y_1, z_1)\}$ has a weight of 115. So the Greedy-MAX cannot get an optimal answer in this example.

(d)

We can use **Theorem 1** to prove the inequality.

Firstly, we define the current independent system $(E, \mathcal{I})$ which is defined in (a). Each $A \subset \mathcal{I}$ is a disjoint matching on X, Y, Z. Then we define another three independent systems: $(E, \mathcal{I}_X)$, $(E, \mathcal{I}_Y)$ and $(E, \mathcal{I}_Z)$, and their definition are as follows:

- $(E, \mathcal{I}_X)$: $A \in \mathcal{I}_X$ if and only if : for any $(x_i, y_i, z_i), (x_j, y_j, z_j) \in \mathcal{I}_X$, $x_i \neq x_j (i \neq j)$.
- $(E, \mathcal{I}_Y)$: $A \in \mathcal{I}_Y$ if and only if : for any $(x_i, y_i, z_i), (x_j, y_j, z_j) \in \mathcal{I}_X$, $y_i \neq y_j (i \neq j)$.
- $(E, \mathcal{I}_Z)$: $A \in \mathcal{I}_Z$ if and only if : for any $(x_i, y_i, z_i), (x_j, y_j, z_j) \in \mathcal{I}_X$, $z_i \neq z_j (i \neq j)$.

Then we prove that $(E, \mathcal{I}_X)$ is a matroid, and $(E, \mathcal{I}_Y)$, $(E, \mathcal{I}_Z)$ can be proved in the same way.

- **Hereditary** For any $A \in \mathcal{I}_X, B \subset A$, since any mathches $(x, y, z), (x_0, y_0, z_0) \in B$, we have $(x, y, z), (x_0, y_0, z_0) \in A$. So $x \neq x_0$, then $B \in \mathcal{I}_X$.

- **Exchange Property** For any $A, B \in \mathcal{I}_X$, $|A| < |B|$. Since each triple in A has different x, let A_X be the set of x in A, we can get $|A_X| = |A|$, samely we have: $|B_X| = |B|$, so $|B_X| > |A_X|$, then there exists one triple $(x_0, y_0, z_0) \in B$, $x_0 \notin A_X$. So $A' = \{(x_0, y_0, z_0)\} \cup A \in \mathcal{I}_X$.

Considering the following equation and **Theorem 1**:

$$\mathcal{I} = \mathcal{I}_X \cap \mathcal{I}_Y \cap \mathcal{I}_Z$$

So we have: $\max\limits_{F \subseteq D} \frac{v(F)}{u(F)} \leq 3$.

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 $\text{OPT}(i, b, c)$ = 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 $\text{OPT}(i, b, c)$.

   (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 of Problem 3

(a)

Since $OPT(i, b, c)$ tries to maximize value of $\{p_1, p_2, ..., p_i\}$, with the restrictions of c which give the ceiling of number of people choosen. b represent the people already choosen in $\{p_{i+1}, ..., p_n\}$, which restrict our choise of the next person when operating the recursion.

Optiaml substructure:

- **Case 1**: p_i is not selected.

  Since p_i is not selected, choosing from $\{p_1, p_2, ..., p_i\}$ is equivlatent to choosing from $\{p_1, p_2, ..., p_{i-1}\}$. We can refer to the last recurrence for maximal value. So we have $OPT(i, b, c) = OPT(i - 1, b, c)$ in this case. It should be noted that when $c_i < b$, we must refer to this case.

- **Case 2**: p_i is selected.

  When p_i is selected, we should change b and c when move from $i - 1$ to i. Also we add the value by v_i, so we actually have: $OPT(i, b, c) = OPT(i - 1, b + 1, min\{c - 1, c_i - b\}) + v_i$. When this case happens, b_i must be greater than b.

Considering the two cases above, we have the recurrence relationship:

$$OPT(i, b, c) = \begin{cases} 0 & i = 0 \ or \ c = 0 \\ OPT(i - 1, b, c) & c_i < b, i \geq 1 \\ max\{OPT(i - 1, b, c), OPT(i - 1, b + 1, min\{c - 1, c_i - b\}) + v_i\} & c_i \geq b, i \geq 1 \end{cases}$$

The final optiaml answer can be represented as $OPT(n, 0, n)$.
(b) **Pseudo code for the algorithm above**(non-recursion type):

Input: Value list $v = [v_1, v_2, ..., v_n]$, restriction list $[c_1, c_2, ..., c_n]$.
Output: The maximum values we can get.

Construct a 3-Dimensional table OPT with a size of $(n+1) \times (n+1) \times (n+1)$.
Set all value of table OPT to 0.
for i = *1 to n* **do**
  for j = *1 to n - i + 1* **do**
    for k = *1 to n* **do**
      if $c[i] < b[i]$ **then**
        $OPT[i][j][k] = OPT[i-1][j][k]$
      else
        $OPT[i][j][k] = max\{OPT[i-1][j][k], v[i] + OPT[i-1][j+1][min\{c[i]-j, k-1\}]\}$
return $OPT[n][0][n]$

---

* The second-level loop starts from 1 to $n - i + 1$ since the b in $OPT(i, b, c)$ will at most be increased by 1 in one move forward.
(c) The time and space complexity:

- **Time Complexity**: the time complexity should be:

$$\sum_{i=1}^{n} \sum_{j=1}^{n-i+1} \sum_{k=1}^{n} 1 = O(n^3)$$

- **Space Complexity**: Since we just use the 3-Dimensional table to save the value without any other intermediate areas. We have the space complexity is $O(n^3)$.

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