Lab04-Matroid

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

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

∗ Name: Yulong Hui Student ID:518030910059 Email: qinchuanhuiyulong@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.

(a) Assume that I is a subset of E, and $\mathbf{C}$ is the collection set of I. If the out-degree at any node in I is no greater than the constraint, then we call the I set is an independent set and $(S, \mathbf{C})$ is an independent system.

Prove. If $A \subseteq B$ and $B \in \mathbf{C}$, then the node in A must have less out-degree than that in B. So, it is hereditary.

Then if $F, D \in \mathbf{C}$ and $|F| < |D|$, there must be at least one node (assume it is n) satisfying that the out-degree of n in F is less than that in D. Then we can add a edge e ($e \in D$ but $e \notin F$) whose source is n to set F. And now in $F \cup \{e\}$, the out-degree of n is not greater than that in D, so it is still not greater than the constaint, and set F is still in $\mathbf{C}$.

Therefore, $(S, \mathbf{C})$ is a matroid.

(b)The detailed implemention will be explained in (c).

Algorithm 1: Greedy-MAX1

Input: The graph $G = (V, E)$, $w(e)$ and $f(u)$

Output: A correct subset of E

1 Sort the elements in E by weight, so that $w(e_1) > w(e_2) > \cdots > w(e_n)$;

2 $A \leftarrow \emptyset$;

3 **for** $i \leftarrow 1$ **to** n **do**

4   **if** $A \cup e_i$ *satisfies the constraint on out-degree* **then**

5     $A = A \cup e_i$

6 return A;

(c) First, the time complexity of the sorting process is $O(E \log E)$. If we assume the time complexity from line4 to line5 is $f(E, V)$, the total complexity should be $O(E \log E + Ef(E, V))$.Then analyze $f(E, V)$.

To decrease the time complexity and finish the algorithm correctly, we may use some extra space. We can create a link-list to store the edges and an array of V elements to record the current out-degree of every node. Then, every comparing betwenn the current out-degree and constraint (line4) will take $O(1)$. Increasing the out-degree and adding the edge to the link-list(line5) will also take $O(1)$. So the complexity $f(E, V)$ can be $O(1)$, then the total 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. (a) Let $S = X \times Y \times Z$, and $\mathbf{C}$ is a collection of some certain sets. $A \in \mathbf{C}$ iff the triples in A are all disjoint each other, then A is an independent set and (S,$\mathbf{C}$) is an independent system. (As for the proof, it is obvious that if we move a triple out others will still remain disjoint).

(b)

Algorithm 2: Greedy-MAX2

Input: The n triples, $c(i)$

Output: A correct collection $\mathcal{F}$

1 Sort the elements in E by weight, so that $c(tri_1) > c(tri_2) > \cdots > c(tri_n)$;
2 $\mathcal{F} \leftarrow \emptyset$;
3 **for** $j \leftarrow 1$ **to** n **do**
4   **if** $\mathcal{F} \cup e_i$ *is compatible* **then**
5         $\mathcal{F} = \mathcal{F} \cup tri_j$
6 return $\mathcal{F}$;

About more detailed implemention. We can create three hash-tables to record the existing x, y,and z value. Then create a link-list to store the compatible triples. In line4, we will check wether new x_j has been in the X-hash-table, and then the same to y_j, z_j. If compatible, we will add the new value to hash-table and add the triple to the storing list (line5).

(c) We can let $X = Y = Z1, 2$. Let the weight for (1,2,1) is 7, while (1,1,2) weighs 6 and (2,2,1) weighs 5. And let all other triples weight 0. Then, it is easy to see that the Greedy result weights 7, while the optimal one weighs 11.

(d) Let $E = X \times Y \times Z$. And there are three matroids: $(E, I_X), (E, I_Y), (E, I_Z)$, which means if $B \in I_X$, then every triple in B has different x-value.

If we delete some triples in B and form a subset C, then all triples in C still have different x-values, which prove it is hereditary. If $|D| < |B|$, then there must be less x-values in D, we can just choose one triple t whose x-value is not in D but in B, then $D \cup \{t\}$ still have different x-values. So the exchange property is proved.

If $A_1 \in I_X, A_2 \in I_Y, A_3 \in I_Z$, and $A_1 = A_2 = A_3 = A \in \bigcap_{i=1}^{3} \mathcal{I}_i$, then all triples in A have different x,y,z-values so they are disjoint, which meets the requirement.

Then $\mathcal{I} = \bigcap_{i=1}^{3} \mathcal{I}_i = \mathcal{F}$. Then $\max\limits_{F \subseteq E} \frac{v(F)}{u(F)} \leq k = 3$. Therefore, the proposition is proved.

$\square$

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) To compute the $OPT(i, b, c)$, we should discuss some different cases. If $i = 0$, then OPT is 0. If $c = 0$, we can not add the p_i to the team, so the $OPT(i, b, c) = OPT(i-1, b, c)$.

About others, we need to discuss two cases.

• Case 1, OPT selects person-i.

Let $c'_i = min\{c-1, c_i - b\}$, and set c'_i as new c.

Add the v_i, and selects best using {1,2,...,i-1} with this new c-limit.

• Case 2, OPT does not select person-i

OPT selects best using {1,2,...,i-1} with c-limit.

Then we can get:

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

(b) The code is presented in the next page.

Algorithm 3: Crowd Sourcing

Input: n, $\{v_1, v_2, \ldots, v_n\}$, $\{c_1, c_2, \ldots, c_n\}$
Output: maximized $\sum_i v_i$

```
1  Create n^3 int-space f[i, b, c] to store the result, and make them empty at first.
2  Def opt(i, b, c):
3  opt(i, b, c){
4  if f[i, b, c] ≠ empty then
5      return f[i, b, c];
6  if i = 0 or c = 0 then
7      f[i, b, c] ← 0
8  else
9      if c_i < b then
10         f[i, b, c] ← opt(i − 1, b, c);
11     else
12         c' ← min{c_i − b, c − 1};
13         f[i, b, c] ← max{opt(i − 1, b, c), v_i + opt(i − 1, b + 1, c')};
14 return f[i, b, c];
15 }
16 output opt(n, 0, n);
```

(c) The space complexity is obviously $O(n^3)$.

Then about the time complexity: In line1, we should empty the space, whose time-complexity is due to the datasture. And we assume it is $g(n)$ ($g(n) \leq O(n^3)$)

Beacuase calculating the $opt(n, 0, n)$ needn't to solve all elements in $f[i, b, c]$, we just use the memorization method instead of bottom-up one. By doing this, however, the time complexity will still be expressed as $O(n^3)$.

Therefore, both the time and space complexities are $O(n^3)$.

□

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