Lab04-Matroid

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

∗ If there is any problem, please contact TA Yiming Liu.
∗ Name:Zehao Wang   Student ID:518021910976   Email: davidwang200099@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) Define sets of edges $S \subset E$, whose out-degree at any node is not more than $f(\cdot)$, as independent sets, and $\mathbb{C}$ as a collection of such S.

Proof.

- For any set of edges B and each of its subset A, The number of edges in A is not more than B. Therefore the out-degree of any node due to the edges in A is not more than that due to B. Therefore the heredity is satisfied.
- Assume that there does not exist $A, B \subseteq \mathbb{C}$ $x \in B \setminus A$,$|A| < |B|$, $A \cup \{x\} \in \mathbb{C}$.
  Because $|A| < |B|$, therefore $|B \setminus A| \geq 0$.
  Therefore
  $$\forall x \in B \setminus A, A \cup \{x\} \notin \mathbb{C} \tag{1}$$
  Define $d_A(k)$ to be the out-degree of the k-th node in V due to the edges in A and $d_B(k)$ to be the out-degree of the k-th node in V due to the edges in B.
  Because $B \in \mathbb{C}$, therefore
  $$\sum_{k=1}^{|V|} d_B(k) \leq \sum_{k=1}^{|V|} f(v_k) \tag{2}$$
  According to (1), we can know that no edges in B can promise that the out-degree of each node in V is less than $f(\cdot)$.Therefore $\forall v_k \in V$,$d_A(k) = f(v_k)$.
  Therefore
  $$\sum_{k=1}^{|V|} d_A(k) = \sum_{k=1}^{|V|} f(v_k) \tag{3}$$
  Because each edge can make a contribution of only one out-degree, therefore $|A| = \sum_{k=1}^{|V|} f(v_k)$.
  According to (2) and (3), $|B| \leq \sum_{k=1}^{|V|} f(v_k) < |A|$, which is contradictory to the assumption that $|A| < |B|$.
  Therefore, $\exists x \in B \setminus A$, $A \cup \{x\} \in \mathbb{C}$, which means the exchange property is satisfied.

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

(b) Maintain a set of edges A and initialize it to be ϕ.

First sort the edges in E by weight in non-increasing order, then examine the edges from the first to the last.

If the newly-examined edge will not break the constraint on out-degrees, then add it to A.

Else discard it and go on to the next one.

```
Algorithm 1: getMaxWeight($E[1,\cdots,n_e], n_e, n_v, f[1,\cdots,n_v]$)
  Input: $E[1,\cdots,n_e], n_e, n_v, f[1,\cdots,n_v]$
  Output: Set of edges $A$, which has the maximal weight.
1 Sort $E[1,\cdots,n_e]$ by weight in non-increasing order.
2 $A \leftarrow \phi$;
3 $outDegree[1,\cdots,n_v] \leftarrow \{0\}$;
4 for $i = 1$ to $n_e$ do
5     if $outDegree[rank\ of\ startpoint\ of\ E[i]] \leq f[i]$ then
6         $A \leftarrow A \cup \{E[i]\}$;
7 return $A$;
```

(c) The sort has a time complexity of $O(|E| \log |E|)$.

The for-loop should be executed $|E|$ times, and each loop has a complexity of $O(1)$ to check the out-degree, which is $O(|E|)$ in total.

Therefore the time complexity of this algorithm is $O(|E| \log |E|)$.

The space complexity of this algorithm is $O(|V| + |E|)$.

□

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) Define sets of triples S, any two members of which are disjoint, to be independent sets.

(b) For any given set of disjoint triples S, sort its members by weight in non-increasing order.

Maintain a set A and initialize it to be ϕ.

Examine the members in S from the first to the last. If the newly-examined number is disjoint with all the members in A, then add it to A.

Else discard it and go on to examine the next one.

```
Algorithm 2: MAX3DM($E[1,\cdots,n_e], n_e, n_v, f[1,\cdots,n_v]$)
  Input: $triple[1,\cdots,n], n, weight[1,\cdots,n]$
  Output: Set of disjoint triples $A$, which has the maximal weight.
1 Sort $triple[1,\cdots,n]$ by weight in non-increasing order.
2 $A \leftarrow \phi$;
3 for $i = 1$ to $n$ do
4     if $triple[i]$ is disjoint with all members in $A$ then
5         $A \leftarrow A \cup \{triple[i]\}$;
6 return $A$;
```

(c) Considering a set S=(x_0, y_0, z_0),(x_0, y_1, z_1),(x_1, y_0, z_2),(x_2, y_2, z_0) and the weight of (x_0, y_0, z_0) is 2 and that of the others are 1.

Greedy-MAX will choose (x_0, y_0, z_0) only and get a total weight of 2.

But the optimal way is choosing the other 3 and get a total weight of 3.

This is a counterexample to show Greedy-MAX is not the optimal algorithm.

(d) Define S_x, S_y, S_z to be sets of triples any two members of which have different x's,y's and z's separately.$\mathcal{I}_x, \mathcal{I}_y, \mathcal{I}_z$ are the collection of S_x, S_y and S_z.Then we can prove $(S_x, \mathcal{I}_x)$, $(S_y, \mathcal{I}_y)$ and $(S_z, \mathcal{I}_z)$ are all matroids.

- For $(S_x, \mathcal{I}_x)$, obviously its heredity is satisfied.
- For two sets $A, B \in \mathcal{I}_x$ $(|A| < |B|)$, there are at most $|A|$ elements in B which may have the same x as different elements in A. Some of the $|A|$ elements are in $B \cap A$ and the others are in $B \setminus A$. Otherwise, there will be 2 or more elements having the same x in B, which is contradictory to the definition of B.
  Therefore the remaining elements do not have the same x as any element in A. Therefore $\exists e \in B \setminus A$, $A \cup \{e\} \in \mathcal{I}_x$. This means the exchange property is satisfied.
- Therefore $(S_x, \mathcal{I}_x)$ is a matroid.
- Similarly, $(S_y, \mathcal{I}_y)$ and $(S_z, \mathcal{I}_z)$ are both matroids, too.
- By definition, the intersection of $\mathcal{I}_x$, $\mathcal{I}_y$, and $\mathcal{I}_z$ is the collection of set S, any two members of which have different x's, y's and z's, which is $\mathcal{I}$.

Therefore $\mathcal{I}$ is an intersection of 3 matroids.

According to Theorem 1, $\max\limits_{F \subseteq D} \frac{v(F)}{u(F)} \leq 3$.

□

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 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)
- If OPT select p_i, it must be satisfied that p_i can cooperate with the already-chosen b people, namely $c_i \geq b$, and there is at least one seat for p_i, namely $c \geq 1$. Then the number of remaining seats is $min\{c_i, c\}$. This time v_i needs to be taken into account.
- OPT can also choose not to chose p_i even if p_i can be chosen. Then whether to choose p_i depends on the contribution.
- If OPT must not select p_i, that is because p_i can not cooperate with the already-chosen b people, namely $c_i < b$.
- In special cases when $i = 0$, OPT has no person to choose. Therefore the answer is 0.

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

(b) The number of group members is at most the c_i of a certain group member.Therefore we can assume the group capacity to be each c_i of these n candidates and find out when we can get the maximal contribution.

Once we assume the group capacity is an arbitrary value, use greedy strategy to choose as many members with greatest contributions into the group and add their contributions together.A member must not be chosen iff his or her c_i is smaller than the capacity.

Then we get n "maximal" value when the capacity is $c_n, c_{n-1}, \cdots, c_1$ separately. Choose the maximum and execute the greedy strategy above again to find out who are the members.

Algorithm 3: Crowdsource($v[1, \cdots, n], c[1, \cdots, n], n$)

```
   Input: $v[1, \cdots, n], c[1, \cdots, n], n$
   Output: Set of people $A$, which has the most contribution.
1  Sort $v[1, \cdots, n]$ in non-decreasing order and make $c[i]$ and $v[i]$ belong to $p_i$.
2  $w[1, \cdots, n]$, each element of which is initialized as 0;
3  $max \leftarrow 0$;
4  $rankMax \leftarrow 0$;
5  for $i = n$ to 1 do
6      $capacity \leftarrow c[i]$;
7      $j \leftarrow n$;
8      while $capacity > 0$ and $j \geq 1$ do
9          if $c[j] \geq c[i]$ then
10             $w[i] \leftarrow w[i] + v[j]$;
11             $capacity \leftarrow capacity - 1$;
12         $j \leftarrow j - 1$;
13     if $max < w[i]$ then
14         $max \leftarrow w[i]$;
15         $rankMax \leftarrow i$;
16 $capacity \leftarrow c[rankMax]$;
17 $i \leftarrow n$;
18 while $capacity \geq 0$ and $i \geq 1$ do
19     if $c[i] \geq c[rankMax]$ then
20         $A \leftarrow A \cup \{p_i\}$;
21         $capacity \leftarrow capacity - 1$;
22 return $A$;
```

(c) The for-loop from Line 5 to Line 17 has a time complexity of $O(n^2)$.
The sort has a time complexity of $O(n \log n)$.
The for-loop from Line 20 to Line 23 has a time complexity of $O(n)$.
Therefore this algorithm has a time complexity of $O(n^2)$.
This algorithm has a space complexity of $O(n)$.

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