Lab04-Matroid

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

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

∗ Name:Hanzhang Yang   Student ID:518030910022   Email: linqinluli@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 $(E, \mathcal{W})$ and $\mathcal{W}$ is the collection of all edge sets whose every connected node's out-degree is no greater than the constraint.

   The following is the proof on the matroid.

   Hereditary.

   Assume $A \subset B,\ B \in \mathcal{W}$. Since $B \in \mathcal{W}$ and node's out-degree is no greater than $f(u)$, nodes in A have lower out-degree than B.

   Thus $A \in \mathcal{W}$.

   Exchange Property.

   Assume $A, B \in C$ and $|A| < |B|$.

   case 1: There exists a edge $x \in B \backslash A$ and x is not the out edge of node in A.

   Thus $A \cup \{x\}$'s every node's out-degree is still no greater than $f(u)$. $A \cup \{x\} \in C$.

   case 2: All edges in B are the out edges of nodes in A. Thus the graph consists of out edges both in A and B own the same nodes. Threr must be such a node which its out-degree in B is greater than that in A, since $|A| < |B|$.

   Choose a edge x $x \in B \backslash A$ and x is the out edge of the node mentioned before.

   Thus $A \in \mathcal{W}$.

   (b)pseudo cide.

   Algorithm 1: MaximalWeightGrapg

   Input: Graph $G = (V, E)$, weight of edges $W(i)$, constraint of nodes $f(i)$

   Output: a subset of edges with maximal weight, and every node's out-degree $\leq f(i)$

   sort all edges by weight that $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_n)$;
   $R = \emptyset$;
   $F[n] = \{0, 0, \cdots, 0\}$;
   for $i = 1$ *to* n **do**
       if $F[m] < f(m)$ **then**
           $F[m]++$; \\\\e_i is the out edge of node v_m
           add e_i to R ;
       else
           continue;
   Output R

(c) The algorithm consists of a sort and a n-cycle.

The time complexity of the sort is $O(nlogn)$ and the cycle is $O(n)$.

Thus the time complexity of the algorithm is $O(nlogn)$.

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 $(S, \mathcal{C})$, and S is the set of all triples in $X \times Y \times Z$, $\mathcal{C}$ is the collection of all disjoint triples collection. The hereditary is easy to prove.

   (b) pseudo code

   Input: sets X, Y, Z and weight c$(\cdot)$ on all triples in $X \times Y \times Z$
   Output: collection $\mathcal{F}$ of disjoint triples with maximum total weight

   sort all triples by weight that $w(1,1,1) \geq w(1,1,2) \geq \cdots \geq w(x,y,z)$;
   $\mathcal{F} = \emptyset$;
   for $i = 1$ *to* x **do**
       for $j = 1$ *to* y **do**
           for $k = 1$ *to* z **do**
               if *triple*(i,j,k) *is disjoint to all triples in* $\mathcal{F}$ **then**
                   add *triple*(i,j,k) to $\mathcal{F}$;
               else
                   continue;
   Output $\mathcal{F}$

   (c) $X\{1,2\}$ $Y\{3,4\}$ $Z\{5,6\}$

   We give weight $8,7,7,7,7,7,7,1$ to $\{(1,3,5)\ (1,3,6)\ (1,4,5)\ (1,4,6)\ (2,3,5)\ (2,3,6)\ (2,4,5)\ (2,4,6)\}$

   From the algorithm we choose $\{(1,3,5)\ (2,4,6)\}$ as the final set and the weight is 9. However it's clear that $\{(1,3,6)\ (2,4,5)\}$ with weight 14 is optimal.

   (d) **Proof.**

   Assume an independent system $(S, \mathcal{X})$. $\mathcal{X}$ is the collection of all collections of triples in $X \times Y \times Z$ with disjoint $\mathbf{x} \in X$.

Hereditary

Since $A \subset B,\ B \in \mathcal{X}$, it's easy to see that $A \in \mathcal{X}$.

Exchange Property

Since $|A| < |B|$, the values of x in B is more than in A. Thus there must exists a triple $(m, y, z) \in B$ and m haven't exist in $A's$ triples before. $|A| \cup (m, y, z) \in \mathcal{X}$.

Therefore $(S, \mathcal{X})$ is a matroid.

We can get $(S, \mathcal{Y})$ and $(S, \mathcal{Z})$ in the same way.

Then

$$\mathcal{X} \cap \mathcal{Y} \cap \mathcal{Z} = \mathcal{C},$$

for $\mathcal{C}$ is the collection of all collections with disjoint $x,\ y,\ and\ z$.

Finally we can draw the conclusion that $\max\limits_{F \subseteq D} \frac{v(F)}{u(F)} \leq 3$ by theorem 1.

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)When choosing the i_{th} person, there are two cases.
Case 2: $OPT(i, b, c) = max\{OPT(i-1, b, c),\ v_i + OPT(i-1, b+1, min\{c_i - b, c\})\}$

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

(b)**pseudo code:**

Algorithm 2: OPT(i,b,c)

Input: $P\{(v_1, c_1),\ (v_2, c_2), \cdots, (v_n, c_n)\}$
Output: a set of people

$A[n] = \{0, 0, \cdots, 0\}$; \\ used to store the result, not in the recursion
if $i = 0$ **then**
  return 0;
else if $c = 0$ *or* $c_i < b$ **then**
  return $OPT(i-1, b, c)$;
else
  if $OPT(i-1, b, c) \le v_i + OPT(i-1, b+1, min\{c_i - b, c\})$ **then**
    A[i]=1;
    return $v_i + OPT(i-1, b+1, min\{c_i - b, c\})$;
  else
    return $OPT(i-1, b, c)$;
$OPT(n, 0, n)$; \\ the begin of the recursion
Output A;

(c) The deep of the recursion is n.

Space Complexity:$O(n)$

The space complexity is $O(n)$, since every recursion costs constant space.

Time complexity:$O(n^3)$

Consider the worst case, which chooses n people. For the deep of the recursion is n, and in the i_{th} recursion the time cost is 2^{i-1}.

Thus the time complexity is $O(2^n)$.

And using a three-dimensional array to store the result can make the time community become $O(n^3)$ and the space complexity become $O(n^3)$.

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