Lab04-Matroid

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

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

∗ Name: Futao Wei   Student ID: 518021910750   Email: weifutao@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) **Independent:** A set of edges is independent if the out-degree at any node u of the edges in the set is no greater than $f(u)$. Denote $\mathbf{C}$ as:

$$\mathbf{C} = \{F \subseteq E \mid F \text{ is a set of edges satisfying the out-degree constraint}\}$$

To prove that the independent system $(E, \mathbf{C})$ forms a matroid:

- Hereditary property: Given any $F \in \mathbf{C}$ and $P \subseteq F$, P must satisfy the out-degree constraint, that is, $P \in \mathbf{C}$.
- Exchange property: Given $A, B \in \mathbf{C}$ and $|A| < |B|$, it's obvious that

$$\exists x \in E \text{ such that } x \in B \backslash A$$

  There are two cases about edge x:
  Case 1: The start node of x is not in A. In this case, $A \cup \{x\} \in \mathbf{C}$.
  Case 2: The start node of x is in A. Denote it as s. In this case, i. $A \cup \{x\} \in \mathbf{C}$ if the out-degree of s in A is less than $f(s)$. ii. $A \cup \{x\} \notin \mathbf{C}$ if the out-degree of s in A is equal to $f(s)$.
  We'll prove by contradiction that the ii situation in Case 2 cannot hold for every x. If so, we'll arrive at $|A| = |B|$! Thus we've prove the exchange property.

(b)

Algorithm 1: Greedy-MAX

```
Sort edges in E into ordering w(e_1) ≥ w(e_2) ≥ ··· ≥ w(e_m);
A ← ∅;
for i = 1 to m do
    if A ∪ {e_i} ∈ C then
        A ← A ∪ {e_i};
output A;
```

(c) Sorting: $O(m \log m)$. Checking: $O(m)$.
Thus, Greedy-MAX algorithm takes $O(m \log m)$ time.

□

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 $\mathbf{C}$ as

$$\mathbf{C} = \{F \subseteq D \mid \text{any two triples in } F \text{ are disjoint}\}$$

Then $(D, \mathbf{C})$ is an independent system. Proof of hereditary property:
Given $A \subseteq B, B \in \mathbf{C}$, it's obvious that $A \in \mathbf{C}$, since removing some triples from B will not generate joint triples.

(b) Denote $|X| = p, |Y| = q, |Z| = r$.

Algorithm 2: Greedy-MAX

```
Sort triples in D into ordering c(d_1) ≥ c(d_2) ≥ ··· ≥ c(d_pqr);
A ← ∅;
for i = 1 to pqr do
    if A ∪ {d_i} ∈ C then
        A ← A ∪ {d_i};
output A;
```

(c) $X = Y = Z = \{1, 2\}$. $c((1, 1, 1)) = 5, c((1, 2, 2)) = 4, c((2, 1, 1)) = 3$, all other triples have weight 0.
In this case, Greedy-MAX algorithm ends up with $A = \{(1, 1, 1), (2, 2, 2)\}$ with a total weight 5. However, the optimal choice is $A = \{(1, 2, 2), (2, 1, 1)\}$ with a total weight 7.

(d) Define $\mathbf{C_i}, i = 1, 2, 3$ as

$$\mathbf{C_i} = \{F \subseteq D \mid \text{the } i\text{-th elements of triples in } F \text{ are different from one another}\}$$

Let's prove that $(D, \mathbf{C_i}), i = 1, 2, 3$ is a matroid.

- Hereditary property: Same as the proof in (a).
- Exchange property: Given $A, B \in \mathbf{C_i}$ and $|A| < |B|$, it's obvious that

$$\exists d \in B \backslash A \text{ such that } A \cup \{d\} \in \mathbf{C_i}$$

since there must exist a triple in B whose the i-th element is different from those of all the triples in A.

Since $(D, \mathbf{C})$ is the intersection of 3 matroids $(D, \mathbf{C_i}), i = 1, 2, 3$, we have $\max_{F \subseteq D} \frac{v(F)}{u(F)} \leq 3$ according to Theorem 1.

$\square$

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

   (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) Optimal substructure:
Case 1: OPT selects p_i.

- collect contribution v_i
- the number of seats left decreases
- must include optimal solution to problem consisting of remaining $i-1$ people

Case 2: OPT does not select p_i.

- must include optimal solution to problem consisting of remaining $i-1$ people

Recurrence:

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

(b)

Algorithm 3: Memorization

```
for b = 0 to n do
    for c = 0 to n do
        M[0, b, c] = 0;
for i = 0 to n do
    for b = 0 to n do
        M[i, b, 0] = 0;
for i = 1 to n do
    for b = 0 to n do
        for c = 1 to n do
            if c_i < b then
                M[i, b, c] = M[i − 1, b, c];
            else
                M[i, b, c] = max{v_i + M[i − 1, b + 1, min{c − 1, c_i − b}], M[i − 1, b, c]};
return M[n, 0, n];
```

Algorithm 4: Find-Solution(j, b, c)

```
if j = 0 or c = 0 then
    return ∅;
if v_j + M[j − 1, b + 1, min{c − 1, c_j − b}] ≥ M[j − 1, b, c] then
    return {p_j} ∪ Find − Solution(j − 1, b + 1, min{c − 1, c_j − b});
else
    return Find − Solution(j − 1, b, c);
```

(c) Time complexity: $O(n^3) + O(n) = O(n^3)$.
Space complexity: $O(n^3)$.

□

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