CS 577 (Epic) # Assignment 11 – Intractability Spring 2021

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Zhuoyan Xu Wisc id: zxu444

Intractibility

1. *Kleinberg, Jon. Algorithm Design (p. 506, q. 4).* A system has a set of n processes and a set of m resources. At any given point in time, each process specifies a set of resources that it requests to use. Each resource might be requested by many processes at once; but it can only be used by a single process at a time. If a process is allocated all the resources it requests, then it is active; otherwise it is blocked.

   Thus we phrase the Resource Reservation Problem as follows: Given a set of processes and resources, the set of requested resources for each process, and a number k, is it possible to allocate resources to processes so that at least k processes will be active?

   For the following problems, either give a polynomial-time algorithm or prove the problem is NP-complete.

   (a) The general Resource Reservation Problem defined above.

This problem is NP- complete. the set packing problem can be reduced to this problem

① We first show this problem is NP.

for a given set k process, initialize used = {} as empty set.

for i = 1 ... k

add requested resources of process p_i to used set

if there is new added resources already in p_i, k is not a solution. return false

End

return true

End

runs in $O(km^2)$

② We show this problem is NP- complete

By cook's thm, for all $Y \in NP$. $Y \leq_p 3SAT$.

Since $3SAT \leq_p$ Independent set $\leq_p$ Set packing problem

This problem is NP- complete.

(b) The special case of the problem when $k = 2$.

This problem is in P.

We brute-force all possible pair of 2 process $O(k^2)$.

for each pair, we check whether they conflict some resources.

in $O(k^2 m)$.

This can be solved in poly-time

(c) The special case of the problem when there are two types of resources–say, people and equipment– and each process requires at most one resource of each type (In other words, each process requires one specific person and one specific piece of equipment.)

This is in P.

For each resources in $1 \cdots m$. we find its corresponding process. Then we select those process whose 2 resources are met. this requires $O(m)$.

(d) The special case of the problem when each resource is requested by at most two processes.

This is a special case in set packing problem (where each item only occurs at at most 2 sets).

This is also a special case in independent set problem (each edge only connected to 2 nodes).

Independent set problem. can be reduced to this problem.

So this is NP-complete.

2. *Kleinberg, Jon. Algorithm Design (p. 506, q. 7).* The 3-Dimensional Matching Problem is an NP-complete problem defined as follows:

   Given disjoint sets X, Y, and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, does there exist a set of n triples in T that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?

   Since 3-Dimensional Matching is NP-complete, it is natural to expect that the 4-Dimensional Problem is at least as hard.

   Let us define 4-DimensionaI Matching as follows. Given sets W, X, Y, and Z, each of size n, and a collection C of ordered 4-tuples of the form (w_i, x_j, y_k, z_ℓ), do there exist n 4-tuples from C so that no two have an element in common?

   Prove that 4-Dimensional Matching is NP-complete. Hint: use a reduction from 3-Dimensional Matching.

① We first show this problem is NP.

for a given set of n tuples. initialize used = {} as empty set.

for i = 1 ... n in given set

  add used elements $(w^{(i)}, x^{(i)}, y^{(i)}, z^{(i)})$ into use = {}

  if there is new added resources already in used. the set not a solution. return false

  End

return true

END

runs in $O(n^2)$

② We show this problem is NP- complete

for 3-D matching problem, we add 4-th entry to triple. to make triple a tuple. The 4-th entry should be unique (it can be tuple ID, each time we have a new tuple. ID++). Then the problem is reduced to 4-D matching problem in polynomial.

Since 3-D matching is NP- complete, we have 4-D matching is NP- complete.

3. *Kleinberg, Jon. Algorithm Design (p. 507, q. 6).* Consider an instance of the Satisfiability Problem, specified by clauses $C_1, ..., C_k$ over a set of Boolean variables $x_1, ..., x_n$. We say that the instance is monotone if each term in each clause consists of a nonnegated variable; that is, each term is equal to x_i, for some i, rather than $\overline{x_i}$. Monotone instances of Satisfiability are very easy to solve: They are always satisfiable, by setting each variable equal to 1.

   For example, suppose we have the three clauses

   $$(x_1 \vee x_2), (x_1 \vee x_3), (x_2 \vee x_3).$$

   This is monotone, and the assignment that sets all three variables to 1 satisfies all the clauses. But we can observe that this is not the only satisfying assignment; we could also have set x_1 and x_2 to 1, and x_3 to 0. Indeed, for any monotone instance, it is natural to ask how few variables we need to set to 1 in order to satisfy it.

   Given a monotone instance of Satisfiability, together with a number k, the problem of *Monotone Satisfiability with Few True Variables* asks: Is there a satisfying assignment for the instance in which at most k variables are set to 1? Prove this problem is NP-complete.

NP

① This problem is NP. For each potential solution, we check clause one by one. It can be done in poly time.

② a set cover problem. can be reduced to this problem.

Denote $U_1 \cdots U_n$ element $\in U$ (universe) in SC to $C_1 \cdots C_k$ clauses

Denote subset of U : $S_1 \cdots S_n$ in SC to $x_1 \cdots x_n$ terms

If U_i is in S_j, we say C_i contains x_j.

U_1 is in S_1, S_2, S_3. → $C_1 = (x_1 \vee x_2 \vee x_3)$

If we have $\geq m$ solutions to this problem.

we solve SC subsets. problem.

This problem is NP-complete.

4. *Kleinberg, Jon. Algorithm Design (p. 509, q. 10).* Your friends at WebExodus have recently been doing some consulting work for companies that maintain large, publicly accessible Web sites and they've come across the following Strategic Advertising Problem.

   A company comes to them with the map of a Web site, which we'll model as a directed graph $G = (V, E)$. The company also provides a set of t trails typically followed by users of the site; we'll model these trails as directed paths $P_1, P_2, ..., P_t$ in the graph G (i.e., each P_i is a path in G).

   The company wants WebExodus to answer the following question for them: Given G, the paths $\{P_i\}$, and a number k, is it possible to place advertisements on at most k of the nodes in G, so that each path P_i includes at least one node containing an advertisement? We'll call this the Strategic Advertising Problem, with input $G, \{P_i : i = 1, ..., t\}$, and k. Your friends figure that a good algorithm for this will make them all rich; unfortunately, things are never quite this simple.

   (a) Prove that Strategic Advertising is NP-Complete.

① First we prove this NP. given G. $\{P_i\}$ and node set N $X_1 \cdots X_k \in N$

- for each path $P_i \in \{P_i\}$. check whether there exist one node in N. If there is, break to next P_i.
  If no nodes in P_i in N. return false.

- After checking. all P_i without return false. return true.
  complexity: $O(tk|V|)$

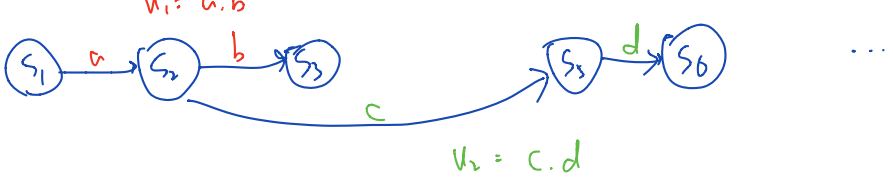
② Set cover problem can be reduced to this problem.

set $U_1 \cdots U_t$ element $\in U$ (universe) to SC to $P_1 \cdots P_t$ paths

Set subset of $U : S_1 \cdots S_N$ in SC to $X_1 \cdots X_k$ nodes

If U_i is in S_j. we say P_i go through X_j

$U_1 \in S_1, S_2, S_3$ $U_2 \in S_2, S_5, S_6, \cdots$



Then we reduce to set cover problem to this problem.

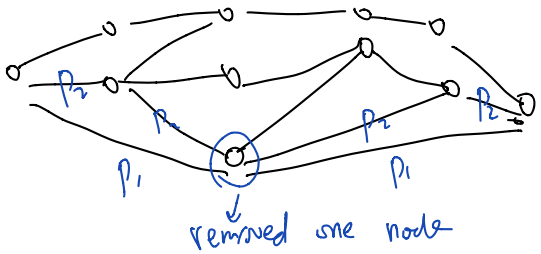
This problem is NP-complete.

(b) Your friends at WebExodus forge ahead and write a pretty fast algorithm S that produces yes/no answers to arbitrary instances of the Strategic Advertising Problem. You may assume that the algorithm S is always correct.

Using the algorithm S as a black box, design an algorithm that takes input $G, \{P_i : i = 1, ..., t\}$, and k as in part (a), and does one of the following two things:

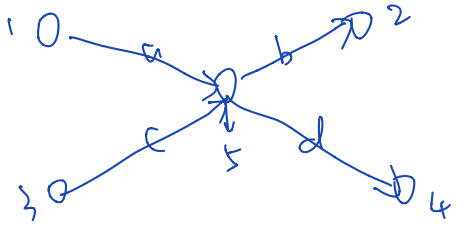
- Outputs a set of at most k nodes in G so that each path P_i inclujdes at least one of these nodes.
- Outputs (correctly) that no such set of at most k nodes exists.

Your algorithm should use at most polynomial number of steps, together with at most polynomial number of calls to the algorithm S.



Initialize solution set $M = \{\}$.

- First we run S on $G, \{P_i\}, k$, we have yes.
- We remove one node. update V'. E' $\{P_i\}$ accordingly.



$P_1 = a \cup d$

$P_2 = c \cup b$

remove (5). $V' = V \setminus \{5\}$.

$E' = E \setminus \{a, b, c, d\} \cup \{1 \to 4, 1 \to 2, 3 \to 2, 3 \to 4\}$.

$P_1 = 1 \to 4$. $P_2 = 3 \to 2$

Then we run S on $G', \{P_i\}, k$.

- If yes, we continue remove nodes repeatly
- If no, we add removed node to solution set M

run S on $G', \{P_i\}, k-1$

If in some iteration, P_i only has 2 nodes and we remove one of them. set $\{P_i'\}_1^{t-1} = \{P_i'\}_1^t \setminus P_i$