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: Gurubharan Kumar Wisc id: kumar76@wisc.edu

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 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.

The 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 = {} an 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
```

2) We show this problem is NP-complete.

By Cook's theorem, ∀ Y ∈ NP. Y ≤P 3SAT

Since 3SAT ≤P independent set ≤P set packing algorithm,

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 set
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 in each item only occurs 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-Dimensional 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 such that each element of $W \cup Y \cup X \cup Z$ appears in exactly one of these 4-tuples?

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 used = {}.
if there is new added element already in used the set
not a solution. return false.
end
return true. Run in $O(n^4)$
end.

2) We show this problem is NP-complete
3-D matching problem, we add 4th entry to triple. to make triple a tuple. The 4th ~~entry~~ entry should have a new tuple. $\in D + t$. 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 $C_1, \ldots, C_m$ over a set of Boolean variables $x_1, \ldots, x_n$. 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.

① This problem is NP. For each potential solution we check one by one. It can be done in poly time
② a set cover problem can be reduced to this problem.

Denote $u_1 \ldots u_2$ elenet $\in$ U(universe) in SC to $c_1 \ldots c_n$ clauses. Denote subset of $U = S_1 \ldots S_n$ in SC to $x_1 \ldots x_n$ terms

If u_1 is in s_j, we say c_i contains x_j.

u_1 is in $s_1, s_2, s_3 \rightarrow c_1 = (x_1 \cup x_2 \cup x_3)$.

If we have $\neq$ on solutions to this problem. we solve SC subsets problem.

The 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.

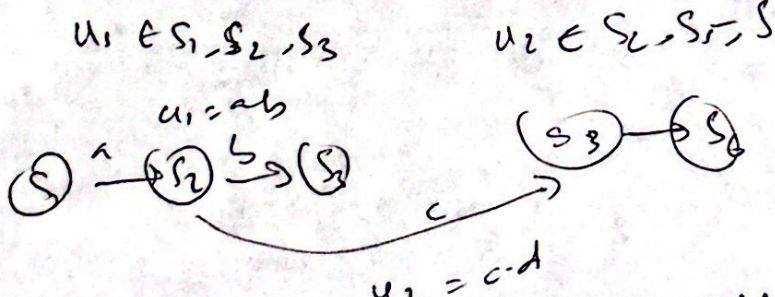
   A company comes to them with the map of a Web site, which we'll model as a directed graph $G = (V, E)$. The company also provides a set of t trails typically followed by users of the site; we'll model these trails as directed paths $P_1, P_2, ..., P_t$ in the graph G (i.e., each P_i is a path in G).

   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 $x_1 \cdots x_k \subseteq V$.
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) in SC to $P_1 \cdots P_t$ paths. set subset of $u = S_1 .. S_n$ in SC to $x_1 \cdots x_k$ nodes. If U_i is in S_j. we say P_i go through x_j.

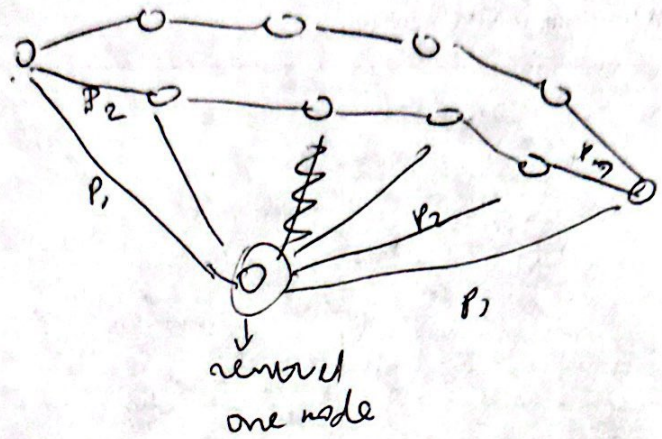


Then we reduce to set cover problem to this problem. This problem is NP-complete.

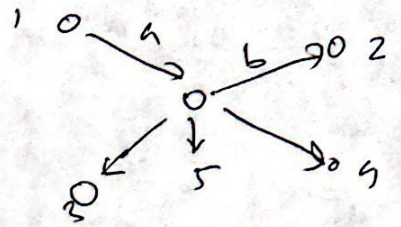
(b) Your friends at WebExodus forge ahead and write a pretty fast algorithm S that produces yes/no answers to arbitrary instances of the Strategic Advertising Problem. You may assume that the algorithm S is always correct.
Using the algorithm S as a black box, design an algorithm that takes input $G, \{P_i : i = 1, \ldots, t\}$, 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 includes 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 $G\{P_i\}_{i=1}^{t}, k$, we have yes. we remove one update $V, E'\{P_i\}$ accordingly.



$P_1 = a \cup d$
$P_2 = c \vee b$
remove 0. $V' = V \setminus \{5\}$
$E' = E \setminus \{a, b, c, d\} \vee \{\}$
$\{1 \to 4, 1 \to 2, 3 \to 2, 3 \to 4\}$

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

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 $G' \cdot \{P_i\} \cdot k-1$

If in some iteration, P_i only has only 2 nodes and we remove one of them. Set $\{P'\}_1^{t-1} = \{P_0, \{1, b\} \mid P_i$

5. *Kleinberg, Jon. Algorithm Design (p. 512, q. 14)* We've seen the Interval Scheduling Problem several times now, in different variations. Here we'll consider a computationally much harder version we'll call *Multiple Interval Scheduling*. As before, you have a processor that is available to run jobs over some period of time.

People submit jobs to run on the processor. The processor can only work on one job at any single point in time. Jobs in this model, however, are more complicated than we've seen before. Each job requires a *set* of intervals of time during which it needs to use the processor. For example, a single job could require the processor from 10am to 11am and again from 2pm to 3pm. If you accept the job, it ties up your processor during those two hours, but you could still accept jobs that need time between 11am and 2pm.

You are given a set of n jobs, each specified by a set of time intervals. For a given number k, is it possible to accept at least k of the jobs so that no two accepted jobs overlap in time?

Show that Multiple Interval Scheduling is NP-Complete.

① The MIS is NP

Given a set of jobs $\{j_i\}$. Iterate all jobs and check the time intervals. If encounter some time intervals twice, return false.

② Independant set $\leq_P$ MIS.

For each node in Is corresponds to job in MIS.

each edge in Is corresponds to time interval in paths.

For all edges $\{e_i\}$ adjacent to node u_j

let job j_i in MIS require time interval $\{t_i\}$ in $\{e_i\}$.

6. *Kleinberg, Jon. Algorithm Design (p. 519, q. 28)* Consider this version of the Independent Set Problem. You are given an undirected graph G and an integer k. We will call a set of nodes I "strongly independent" if, for any two nodes $v, u \in I$, the edge (v, u) is not present in G, and neither is there a path of two edges from u to v, that is, there is no node w such that both (v, w) and (u, w) are present in G. The Strongly Independent Set problem is to decide whether G has a strongly independent set of size at least k.

Show that the Strongly Independent Set Problem is NP-Complete.

① SIS in NP
Given a set of nodes L, use BFS check whether there is any other $\in I$ are close to this node, if there is, return false.

② Independat set $\leq_P$ SIS
consider a graph G in IS,
For each $e_i = (u, v)$ in G. add w_i
so it becomes (u, w_i), (w_i, v)
then add edge between each w_i nodes.

7. *Kleinberg, Jon. Algorithm Design (p. 527, q. 39)* The *Directed Disjoint Paths Problem* is defined as follows: We are given a directed graph G and k pairs of nodes $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. The problem is to decide whether there exist node-disjoint paths $P_1, \ldots, P_k$ so that P_i goes from s_i to t_i.
Show that Directed Disjoint Paths is NP-Complete.

① First prove its NP.
Given set $\{P_i\}$, iterate through each nodes, if encounter some node twice, return false

② 3-SAT $\leq_P$ DDP
For each x_i, we have 4 start nodes $s_1^i \ldots s_k^i$
$w_1^i \ldots w_k^i$. $t_1^i \ldots t_k^i$
we have point nodes $P_1^i \ldots P_{L^i}^{i+k}$
Draw $s_a^i \to P_{ca-1}^i \to w_a^i \to t_a^i$ and $s_a^i \to P_{w-2}^i$
$\to v_{n-1}^i \to t_a^i$.
For each clause c_j, create a node s_j and t_j.
if $x_i \in C_j$, draw $s_j \to P_{2j-1}^i \to t_j$.
if $\bar{x}_i \in C_j$ draw $s_j \to P_{2j}^i \to t_j$.

8. *Kleinberg, Jon. Algorithm Design (p. 508, q. 9)* The *Path Selection Problem* may look initially similar to the problem from the question 3. Pay attention to the differences between them! Consider the following situation: You are managing a communications network, modeled by a directed graph G. There are c users who are interested in making use of this network. User i issues a "request" to reserve a specific path P_i in G on which to transmit data.

   You are interested in accepting as many path requests as possible, but if you accept both P_i and P_j, the two paths cannot share any nodes.

   Thus, the Path Selection Problem asks, given a graph G and a set of requested paths $P_1, \ldots, P_c$ (each of which must be a path in G), and given a number k, is it possible to select at least k of the paths such that no two paths selected share any nodes?

   Show that Path Selection is also NP-Complete.

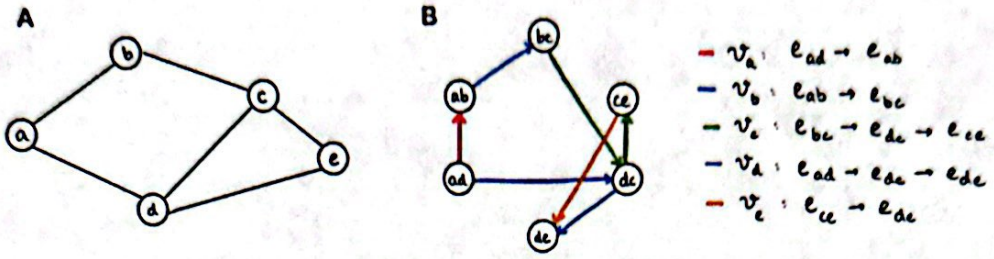


Figure 1: *Path Selection Problem.* A Example directed graph $G = (V, E)$. Observe that the maximum independent set for G has size 2 ($\{a, e\}, \{b, d\}, \{b, e\}$, or $\{a, c\}$). B We build a graph G'. For each edge in G, we create a node in G'. Then for every vertex v_i of G with adjacent edges $e_{j_1}, e_{j_2}, \ldots, e_{j_h(i)}$, we create a path P_i with edges $e_{j_1} \to e_{j_2} \to \ldots e_{j_h(i)}$ picked in some arbitrary order. Observe that G' has a maximum of 2 node-disjoint paths: (P_a:red, P_e:orange), (P_b:blue, P_d:purple), (P_b:blue, P_e:orange), and (P_a:red, P_c:green).

To show that the path selection Problem is NP-complete. we need to demonstrate two things: first, that the problem is in the class NP, and second, that it is NP-hard by reducing another known NP-complete problem to it.

1. Path selection is in NP: To show that the path selection problem is in NP, we need to prove that given a potential solution, we can verify its correctness in polynomial time. In this case, we can easily verify if a given selection of k paths satisfies the requirements of the problem: • check if the selected are distinct. This

can be done by checking for any common nodes between any pair of paths, which can be done in polynomial time. Count the number of selected paths. If the count is at least K, then the the solution is valid.

Path selection is NP-hard: To prove NP-hardness, we can reduce another known NP-complete problem to the path selection problem. In this case, we will reduce the Maximum Independent set (MIS) problem to path selection.

The Maximum Independent set problem: Given a graph G. In this case, we will reduce the maximum Independent set (MIS) problem.

To perform the reduction, we build a graph GO as described in the problem statement:

- For each edge in G, we create a node in GO.
- For every vertex v_i of G with adjacent $e_{j1}, e_{j2} \ldots e_{jh(i)}$, we create a path p_i in GO with edges $e_{j1} \to e_{j2} \to \ldots \to e_{jh(i)}$ picked in some arbitrary order.

Now, we can make the following observation:

- If G has a maximum independent set of size k or more, then we can select k or more paths in GO such that no two paths share any nodes. This is because in the maximum independent set, each selected vertex corresponds to a selected edge in GO, and since no two vertices in the independent set are adjacent
- If we can select k or more path in GO such that no two paths share any nodes, then we can construct an independent set of size k or more in G. By selecting the corresponding vertices in G for the selected edges in GO, we can guarantee that no two vertices in the independent set are adjacent. Therefore, the reduction from the maximum Independent set problem to the path selection problem is valid.