Assignment 11 – Intractability

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: Kayley Seow

Wisc id: kseow

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.

Solution:

Certificate: Guarantee that they don't have overlapping resources.

Certifier: we iterate through the array of k, and if k's resources are not in the array, we add to the array, but if k's resources are in the array, return false.

Prove that it is an NP Problem.

Prove in NP-Hard - use an independent set, where adjacent nodes are not chosen. Nodes were processes, and edges were resources, meaning that if they were connected, they would not be selected, and depending on that we could verify.

Forwards: we can show that independent set solves through previous explanation.

1:1 node to processor

IS $\qquad\qquad\qquad$ RRP

$S = \{n_1, n_2, \dots n_k\} \longrightarrow S' \{P_1, \dots P_k\}$

no two nodes adjacent $\qquad$ they don't share

Backwards: $S = \{n_1, \dots, n_k\} \leftarrow S' \{P_1, \dots P_k\}$ because of one to one, because nodes don't share edges are adjacent

No two processes share the same resource

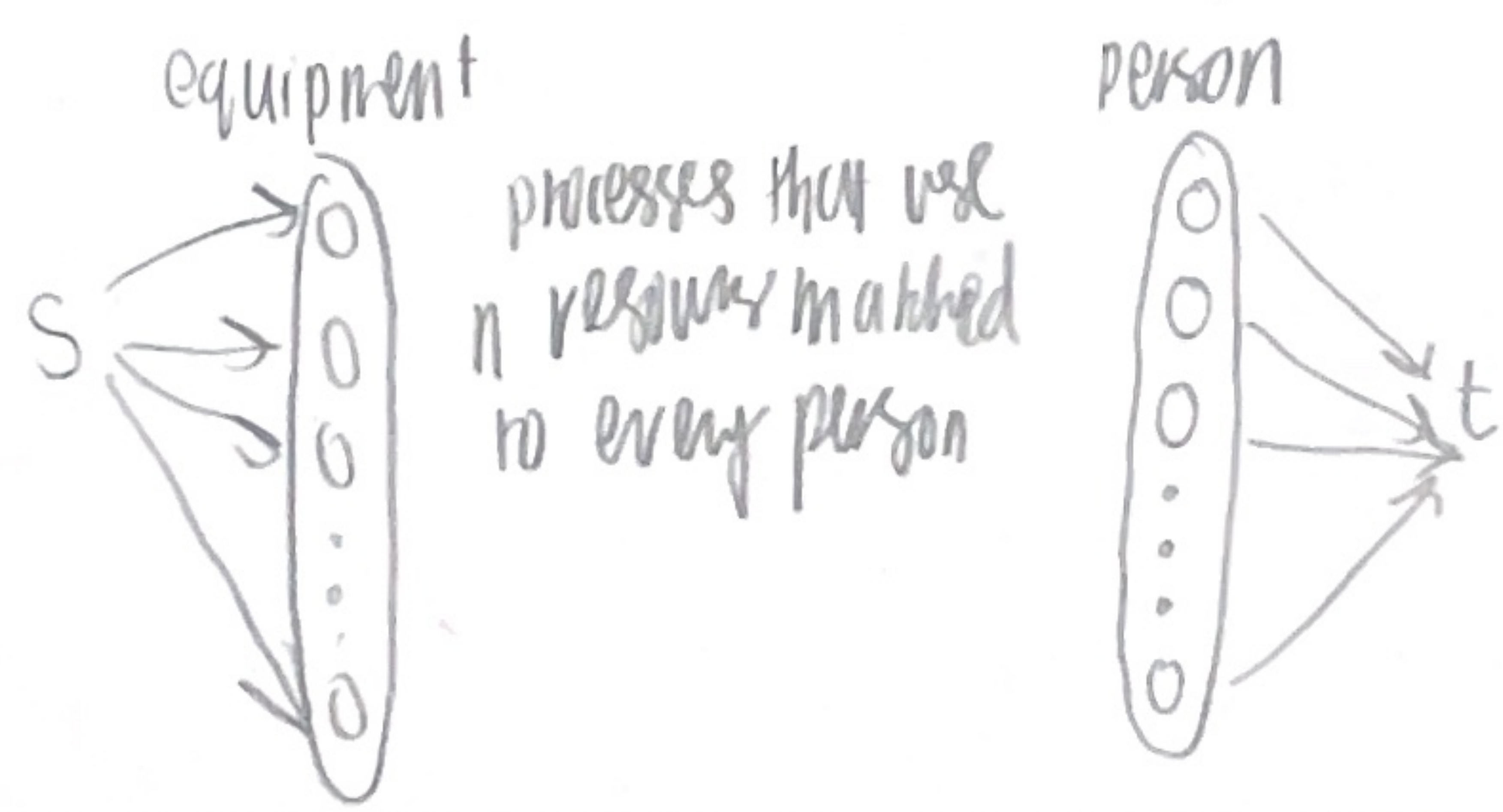
(b) The special case of the problem when $k = 2$.

Solution:
In the case of problem when $k=2$, we would have to do $\binom{n}{2}$, n choose 2 of pairs, and then we would for loop to check through the both edges of process node, to make sure they they are not connected. In the case of m possible resources, we would need the runtime to be $O(n^2 m)$, n^2 based on the nested forloop check. and m for the set of m possible resources.

(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.) algo from prior unit - network flow

person to equipment
bipartite matching

Solution:



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

Solution:
If each resource is a edge, a node a vertex, then having it at most two process would not help. only if it is at most one process. Because this is not making all any simpler. it is just a special case of np complete from the first part of the problem. and does not help with making our problem solving easy.

No clauses overlap
more clauses here
more double tips

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.

Solution:

Certificate: set of n size tuples that satisfy the matching problem.
Certifier/Algorithm: We would loop through the sets to make sure that each element of the disjoint sets X, Y, and Z are contained in the set of n triples in T.
⇒ we have proved algorithm is in NP

With problem reduction, we reduce from the three dimensional problem and use a similar theory behind it.

- Forwards: From a triple to a quad. in our last space we just need to add an extra variable, either from the X, Y, or Z, but always consistent. So, we could have {X, Y, Z, __}, where either X, Y, or Z could be in the graph. With this, we know that the quads would be unique.

{(1,4,7), (2,5,8), (3,6,9)} → {(1,1,4,7), (1,2,4,7), (1,3,4,7), (2,1,5,8), (2,2,5,8), (2,3,5,8), (3,1,6,9), (3,2,6,9), (3,3,6,9)}

check that $T \subseteq X \times X \times Y \times Z$

- Backwards: Knowing that the quads are unique, by removing the duplicate, our 3D conclusion still stands.

⇒ we have proved algorithm is NP Hard, thus NP-Complete

3. *Kleinberg, Jon. Algorithm Design (p. 507, q. 6).* Consider an instance of the Satisfiability Problem, specified by clauses $C_1, ..., C_m$ 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

   n or operations
   n-1 and operations

   $$\overset{\text{or}}{(x_1 \vee x_2)}, \overset{\text{and}}{(x_1 \vee x_3)}, (x_2 \vee x_3).$$

   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.

Solution:

Certificate: A combination of X_1, X_2, and X_3 were inputted into the expression, where we get a 2 as a result.

Certifier/algorithm: We would use the and in the equation, plug in X_1, X_2, and X_3 in order for the expression $(X_1 \vee X_2) \wedge (X_1 \vee X_3) \wedge (X_2 \vee X_3)$. $O(k)$

⇒ proof that this equation in NP

We would use the vertex cover problem in order to solve this problem.

$VC \to SP$

$X_1, \ldots X_n \leftarrow$ nodes

clauses ← edges

reduction: converting graph from vertex cover to this situation, we have our nodes being the Boolean variables, and edges connecting them being a clause that they are in.

Forward: SP returns yes iff VC returns yes. VC touches all edges, all clauses are covered. We can guarantee that it is $\leq k$.

backward: If not VC, it implies not SP. This means that $\leq k$ DNE. If not a VC, not all edges are guaranteed to touch, which means that not all the clauses are covered, which means there is no number less than k.

⇒ Proof that SP is at least NP hard, which makes this 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.

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).

optimization/minimization

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.

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

k=2

gives you solution. verify that it is correct nodes in graph can use to show, and follow the path (nodes = nodes used)

Solution: Input: G, Paths, and the number k.

Certificate: Which k vertices to choose, to hit all the paths in G, set of k nodes that are able to hit all the paths in G, from P_i, i=1 to k.

Certifier/Algorithm: We will iterate through G, for each of the paths we check that they contain at least 1 k node. Number of paths x number of nodes, so we end up with an O(kt) runtime, which is polynomial.

→ With an O(kt) runtime, and the certificate/certifier, we prove that SAP is in NP.

Reduction: Path cover problem, reduced from the vertex cover problem. For each edge, it would mean that we have an edge between the nodes, and the node on our vertex cover would correspond to a node which is contained in the path.

Forward: Our SAP returns yes iff. VC touches all edges, and all the paths are covered, meaning we can guarantee that k, at most k nodes, is valid.

Backward: If not VC, implies not SAP. If not a VC, not all edges are guaranteed to touch, which means that not all the paths will be covered by the nodes, which means that the current k is not enough.

→ Proof that SAP is NP Hard, at least as hard as VC, which due to the reduction, makes this SAP problem NP complete.

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

Using the algorithm $\mathcal{S}$ 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 includes at least one of these nodes.
- Outputs (correctly) that no such set of at most k nodes exists.

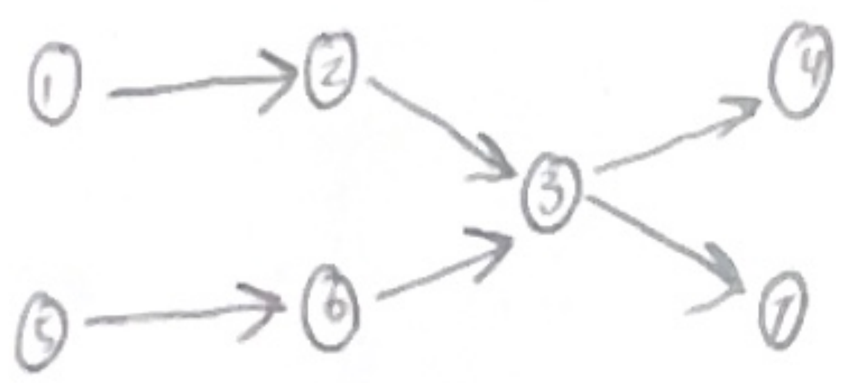
Your algorithm should use at most polynomial number of steps, together with at most polynomial number of calls to the algorithm $\mathcal{S}$.

search to decision reduction.

black box can instantly produce answer for node

Solution:

G



$f(G,1)=Y$

- randomly pick a node to remove, 2 - still returns Y

$f(G^*,1)=Y$

- pick node to remove, 3 - returns N because you need it

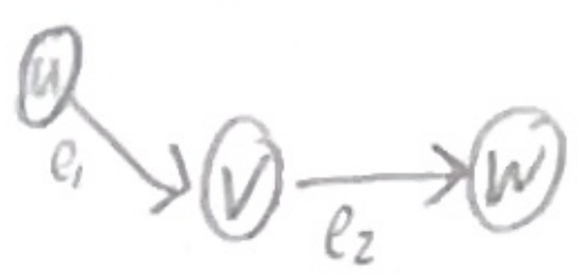
$f(G'',1)=N$

Base Case: We have no more paths, and having no k means we have enough ads, k left over means we need more.

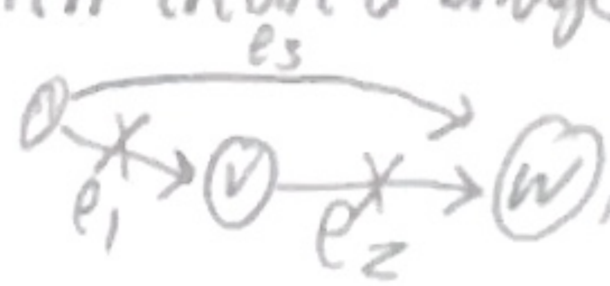
1. We pick a vertex v, at random

2. We remove v, removing the edges related to v, and building bridges to any downstream nodes that v might possibly be connected to.

For example, in the case of



, we would remove edge 1 and edge 2, from u to v and v to w respectively, and then create a "bridge" from u to w, leaving us with



, where u is connected to w, even without v.

3. We call the function f (new Graph, k), where new graph is our graph without vertex v which was removed, and k is the number of nodes. In the case that we have a Y, from algorithm S, we do nothing as it means the edge does not change, but if S returns N, we would put an ad on v as it means that it is an ideal node. In the case of No, we add v to the answer, and remove all paths with v. We decrease k by 1, and then recurse on the new graph, k-1.

Our runtime is $O(k \cdot n \cdot (m+f))$, where k is the number of recursions, n is for going through all of the vertices, and m is for removing the edges of v is connected to, and O(f) arbitrary runtime for calling our function on the next recursive step.