Introduction to Algorithms	Practice questions for Prelim 2
CS 4820, Fall 2019	with Solutions

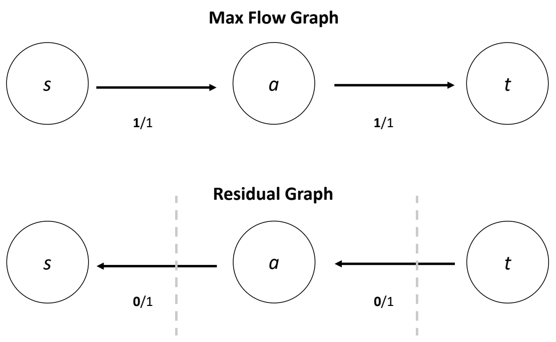
1. *(20 points)* Short answer. Each of these questions asks for a true/false answer or a short answer to a question. Follow the instructions for each question. **No proofs are required in this section.**

a. *(4 points)* **True or false?** Suppose for some flow graph G you have a maximum flow f with flow value $v(f)$ and corresponding residual graph G_f. *Then there exists* ***exactly one*** *(s,t)-cut $\{A, B\}$ such that the capacity $c(A, B)$ of the cut is equal to $v(f)$.*

If false, give a counter-example. If true, give a short (one phrase or one sentence) explanation of why.

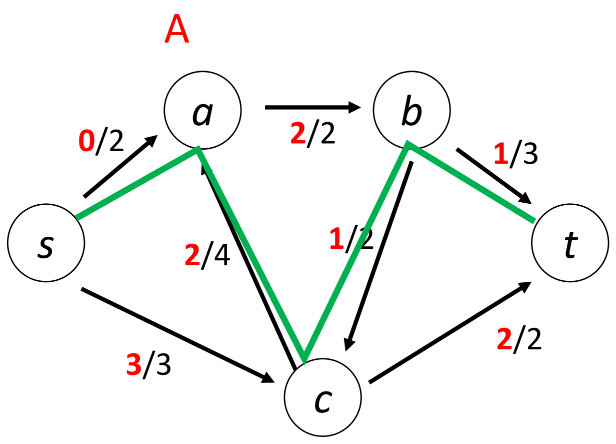
Solution: False. There can be more than one minimum cut for a flow network, and a minimum cut would be a cut on *any* max flow's residual graph.

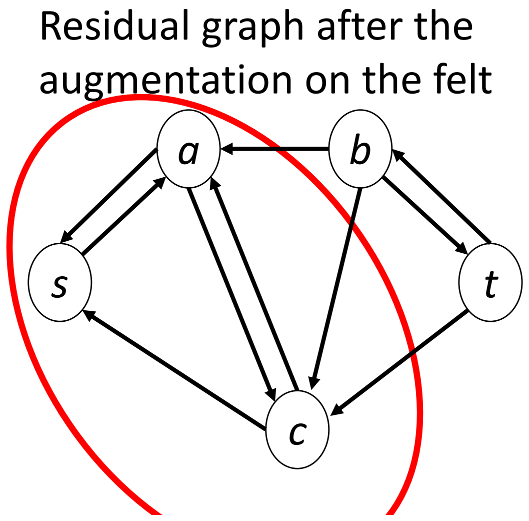
Example: this graph has a single max flow f but two different min cuts.



b. *(4 points)* **Maximum flow?** The following graph presents a flow over a flow network, with each edge e listing the flow f_e in red over the capacity c_e in black as f_e/c_e.

Is this a maximum flow? If not, also (i) give a list of nodes that constitutes a valid augmenting path on the residual graph, and (ii) draw the path on the graph. (We give you a copy of just the vertices for scratch work.)





Solution: This is not a max flow; there is an augmenting path s, a, c, d, b, t marked on the left in green.

c. *(4 points)* **Minimum Cut.** Indicate the minimum capacity (s, t)-cut in the network above and say the cut's value. You can list the nodes on the source side, or circle either the source or the sink side on the figure.

Solution: The source-side of the minimum cut is $A = \{s, a, d, c\}$ with $B = \{b, t\}$ marked on the right in red.

d. *(8 points)* For the following two problems, identify if they are solvable in polynomial time, or NP-complete. For each answer give a short explanation (maximum one sentence).

(i) Given a directed graph $G = (V, E)$ with two nodes $s, t \in V$ as source and sink, is there a simple path (i.e., one with no repeat nodes) from s to t in this graph using **at most** k edges?

Solution: Polynomial time solvable using BFS, or any shortest path with 1 as edge lengths.

(ii) Given a directed graph $G = (V, E)$ with two nodes $s, t \in V$ as source and sink, is there a simple path (i.e., one with no repeat nodes) from s to t in this graph using **at least** k edges?

Solution: NP-complete, the special case when $k = n - 1$ is the Hamiltonian path problem.

2. *(14 points)* The university is considering sending a mandatory survey to undergraduate students to collect student feedback on each course offered by the school. For a constant $k \geq 1$, they want to send the survey to k students in each class. (You may assume every class has at least k students.) However, they want to have each student fill out *no more than two* such surveys. As an input to this problem, you are given a list of n students X, a list of m classes C that each have at least k students, and for each student $i \in X$ a list C_i of the classes taken by student i.

Give a polynomial-time algorithm that finds a valid selection (if one exists) of which students should receive surveys for each class. You **do not need to prove your algorithm correct, but you must analyze its running time**, which should be polynomial in the number of students n, number of classes m, and the total size of the class lists $\sum_{i \in X} |C_i|$.

Solution: Set this up as a flow problem

- nodes are as follows: s, t, students X and classes C.
- For edges (s, i) for all $i \in X$ with capacity 2, edges (i, c) for student i and class $c \in C_i$ with capacity 1, and edges (c, t) for all classes c with capacity k.

Run Ford-Fulkerson's algorithm to find a flow of value km. An integer flow of this form corresponds to the set of representatives desired.

The running time is linear in $\sum_{i \in X} |C_i|$ for setting up the network, which will have $M = n + m + \sum_{i \in S} |C_i| = O(nm)$ edges. Running Ford-Fulkerson will then take $O(Mkm)$ time.

3. *(16 points)* Cornell CIT has noticed that professors are continually having problems with projectors. As a result, they are training a number of students to be "assistant projector technicians" who know how to fix Cornell projectors when they stop working. For a constant $p \geq 1$, they would like to have at least p students trained as assistants in each of the m classes. Any student trained as assistant can serve as assistant in all classes he/she is taking. However, training students requires a 1:1 training session, which is time-consuming to give. CIT would like to limit how many students they have to train to do this.

The Projector Helper problem is as follows: suppose you have a set X of n students who have applied to be technicians. Each student i is enrolled in some subset C_i of the m total classes available. Is it possible to train k or fewer of these n students to ensure that every class has at least p trained students?

Prove that the Projector Helper problem is **NP**-complete. *This should include all steps of an NP-Completeness proof.*

Solution: This problem is in **NP**: our certificate can be the list of k students that permit a valid assignment. For each of the m classes, we can check that there are at least p trained students in the course. For each course, it can naively take time $O(mk)$ to read through each selected student's course list to see if they're enrolled in order to count the total enrolled students.

NP-Hardness via reduction from Set Cover Take a starting Set Cover problem with universe U and sets S_i, with a maximum number of sets k. Take each element in the universe to be a course and each set to correspond to one student, setting the list of courses C_i for student i to the list of elements in the set S_i. We set k to be the same as from the Set Cover problem, and p to be 1. Rewriting the problem this way takes time linear in the original problem (in fact, the only real work is setting p to 1, so if you recycle the U and S_i directly, you could argue this could be done in constant time).

Proof of correctness. Suppose you have k sets in the original problem that form a valid set cover. Choose the students corresponding to these sets. Because we know that the original sets cover every element in the universe, we know the union of these students' classes will cover all course schedules. This produces a set of k students total.

Suppose you have k students in the reduced problem that ensure there is at least one student per course. Choose the sets in the original problem corresponding to these students. For any element in the universe $u \in U$, we know that the student in the course corresponding to u (call them i) will correspond to a set containing u. This implies that every element will be covered by one of these sets. As there are k students in the reduced problem, there will be k sets in the original.

NP-Hardness via reduction from Vertex Cover Consider a starting Vertex Cover problem based on a graph $G = (V, E)$ and a maximum number of vertices k. For each vertex, make one student, and for each edge, make one class. Each student i is enrolled in the courses C_i corresponding to the edges that connect to that student's vertex (implying that every course has two enrolled students). Set p to 1. Rewriting the problem this way takes time $O(|V| + |E|)$.

Proof of correctness. Suppose you have k vertices in the original problem that form a valid vertex cover. Choose the students corresponding to these vertices. Every class j corresponds to an edge e_j in the original graph, which means the student corresponding to the vertex v_i covering e_j in Vertex Cover will be enrolled in course j. Thus, every course will have at least one student enrolled.

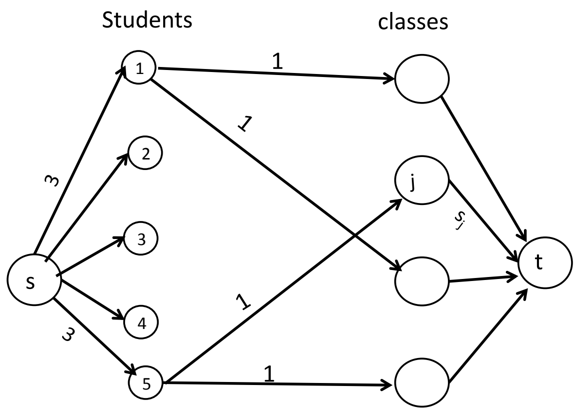
Suppose you have k students in the reduced problem that ensure there is at least one student per course. Choose the vertices in the original problem corresponding to these students. For any edge e_j in the original problem, the student i in course j will correspond to an endpoint of that edge. So, every edge will be covered by one of these k corresponding vertices.

4. Many universities have issues with oversubscribed CS courses. To make things better, and to help ensure class sizes are reasonable, a university that you are helping is planning to switch to the following system. There are n students and k different courses. Each course j has an enrollment size limit s_j. Each student i has a list of CS courses L_i which they would like to take. The university promises all the students that they will be enrolled in exactly three of these courses, if such an assignment is possible.

In this problem, you are asked to give polynomial time algorithms for different versions of this problem. For each part, *you must* ***explain how your algorithm works*** *(for example, explain the meanings of all variables other than loop counters) and* ***analyze its running time, but you do not need to provide a proof of correctness.***

a. Design a polynomial-time algorithm to find the assignment of students to courses as required above, or returns that such an assignment does not exist.

Solution. We set up a max-flow network as shown on the figure



- The network has a source s, sink t, and a node for each student i and each class j.
- Add an edge (s, i) for each student i with capacity 3.
- Add an edge (j, t) with capacity s_j for each class j.
- Add an edge (i, j) with capacity 1 for each student i and class $j \in L_i$.

Run the max flow algorithm to find an integer max flow in this network. The assignment of students to classes exists if and only if the the max flow value is $3n$. The running time of this algorithm is at most $O(nm)$, where $m = \sum_i |L_i|$ the total length of all lists provided by the students. Note that the total capacity of all edges leaving s is $C = 3n$, and the number of edges in this graph is $m + n + k = O(m)$, if we assume each class is listed at least once. Alternately, can bound the number of edges by $O(nk)$, and hence the running time by $O(n^2k)$.

b. If students have more courses in their lists, it is easier to find a desired assignment. In order to encourage the students to do this, we tell them that when a student's list includes more than 3 courses, they can get assigned to up to 4 courses from their list.

   Design a polynomial-time algorithm that is given as input the assignment from part (a) where each student is assigned to exactly 3 courses, and finds an assignment of students to courses where all students are assigned to at least 3 of their courses, and as many as possible are assigned to 4 courses from their list. Your algorithm can change the initial assignment, however it should guarantee that each student is assigned to either 3 or 4 courses from their list, without violating the course capacity constraints.

   Solution 1. Use the same network as above, and find an integer max flow f. By assumption all students are assigned 3 courses, so $f(s,i) = 3$ for all i. Now change the capacity on the edges (s,i) each to 4, and starting with flow f continue running the max flow algorithm.

   The running time is at most $O(mn)$ as above, as the total capacity of the edges leaving s is at most $C = 4n$, and we start with a flow of value $3n$, so will do at most n more augmentations. Observe that in the resulting flow all students will have 3 or 4 classes assigned, as we do not decrease flow on edges leaving the source, and is a maximum flow, so as many students as possible are assigned 4 courses.

   Solution 2. Add a new course x, and put this on all student's list. Corresponds to "not taking a course". Now increase the capacity of the (s,i) edges to 4. We will try all possible capacities $0 \leq s_x \leq n$ for this new course. A flow of value $4n$ exists in this network if and only if there is a way to assign all students to 4 courses, with at most s_x students assigned to the new course x. In terminology of the regular courses, this means that $f(x,t) \leq x_x$ are assigned to 3 courses, and the remaining $n - f(x,t)$ students are assigned to 4 courses. Running time is $O(nmC)$ with m and C as defined in part (a), as we are trying all n different s_x values (or $O(mC \log n)$ if we do binary search for the minimum s_x possible).

   To find the required optimal solution, find the smallest value n_x so that a flow of value $4n$ exists. In this solution $f(x,t) = n_x$, so n_x students are assigned to only 3 real courses, and the remaining $n - n_x$ students are assigned to 4 of the courses from their list.

   Common mistake. A natural alternate solution to think about is just committing to the course assignments computer in part (a), and then using a solution like part (a) with the remaining class capacity to assign as many students as possible to a 4th course. Unfortunately this doesn't work well as demonstrated by the figure. without rearranging the original schedule, one can assign student 2 to a 4th course, but with also rearranging the original schedule, one can find a way to allow students 1 and 2 both to take 4 courses.

   An alternate idea would be to just change the capacity on the (s,i) edges to 4, and use the solution from part (a) on the new network. However, this does not guarantee that all students get at least 3 courses, maybe many will get 4, but some only get 2.

c. Once the new system is in place, the students point out an issue. Some relevant courses are offered in the same time period. So if courses A and B are offered at the same time, clearly any student can only take one of the two. With the assignment above, students were listing only courses in different periods, so they could be assigned to any three of their selected courses. But it would be

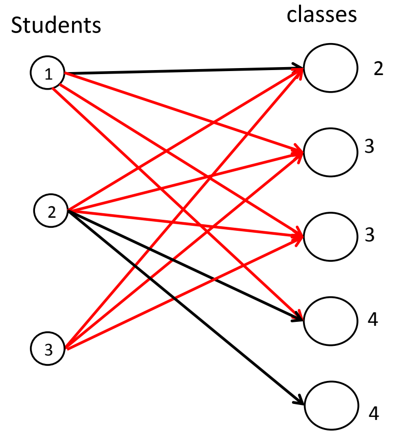


Figure 1: Class capacities are written next to the class nodes. The red edges are the initial assignment of students to classes.

nice to allow them to list courses that conflict. To be precise, we assume courses are in standard course slots. Each course j is in some slot z_j, and no student can take two courses j and ℓ with $s_j = s_\ell$. Suppose we allow students to list courses on their lists L_i that are in the same slots. Design a polynomial-time algorithm to find the assignment to students to 3 courses each from their lists L_i, observing the course capacities, without assigning anyone to two courses in the same time slot.

Solution. We add a new layer of nodes between students are classes. A node (i, p) will correspond to a student in period p. As indicated by the figure. Keep all edges from the source and to the sink as above, but replace the edges (i, j) as follows.

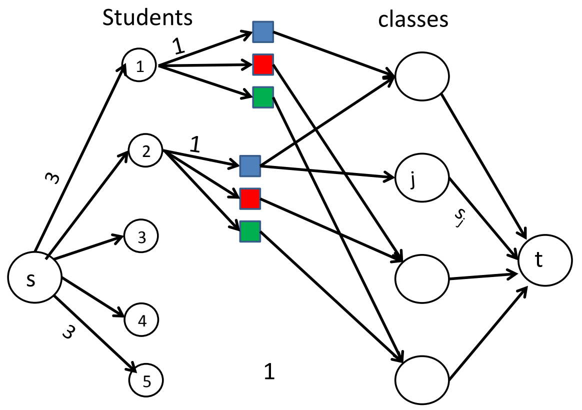


Figure 2: Different colors correspond to different periods.

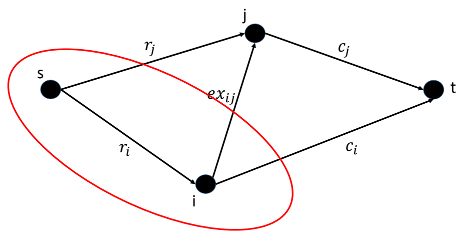
- Add an edge $(i, (i, p))$ with capacity 1 from each student to the new node (i, p) for every period p.

- Add an edge $((i, p), j)$ (with capacity 1) from the new node (i, p) to class j, if $j \in L_i$ and j is offered in period p.

To bound the running time, note that the number of distinct class periods is at most k. Now the number of edges is $n + nk + m + k = O(nk)$. By only adding nodes (i, p) if student i listed a course running in period p, we get the number of edges back to $O(m)$ as above.

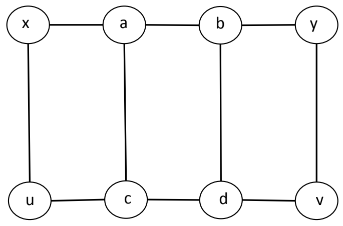
Common mistakes Some students only added nodes for periods p (rather than (i, p) pairs), connected each student to each period with a capacity 1, and then edges from the periods to the classes in these periods. Unfortunately, this way we loose the connection to $j \in L_i$, and may assign students to classes that they didn't list.

5. *(14 points)* You need to select jobs from a set J of jobs. Each job j has a reward r_j if completed, and has a cost for doing the job. For many jobs there are other jobs that make them easier to do, i.e., decrease the cost of doing them by some value $ex_{ij} \geq 0$. Assuming we do all jobs which make job j easier, the cost to do job j is c_j. To be concrete, you have a directed graph G whose nodes are jobs, and edges indicate this relationship. For job j let $P(j)$ denote the set of jobs i with an edge (i, j) in the graph. Then the cost of doing job j assuming we also do a subset of jobs $A(j) \subset P(j)$, equals $c_j + \sum_{i \in P(j) \setminus A(j)} ex_{ij}$. So doing job j is cheapest if you also select to do all of the jobs in $P(j)$. In this case, doing job j costs only c_j. But it's possible to do job j even if none of the jobs in $P(j)$ are done. This will cost as much as $c_j + \sum_{i \in P(j)} ex_{ij}$. Give an algorithm that takes inputs r_j, c_j, and ex_{ij} and finds the set of jobs maximizing rewards minus the cost. You can assume that all input parameters are integers, and your algorithm should run in time polynomial in the number of jobs n, and the total reward $\sum_j r_j$.



Solution. Construct a graph G whose nodes are the jobs with two additional nodes s and t. Two nodes corresponding to jobs i and j are connected with an edge (j, i) with capacity ex_{ij}. There is an edge (s, i) with capacity r_i representing the lost reward, if not doing job i, and an edge (i, t) with capacity c_i. The side of a minimum capacity (s, t)-cut containing the source corresponds to a set of jobs maximizing reward minus cost. (The reasoning is very similar to homework problem 2 of pset 7.) See the figure for an example. It has capacity $\sum_{i \in A} c_i + \sum_{j \notin A} r_j + \sum_{i \in A, j \notin A} ex_{ij}$. If we subtract the fixed sum $\sum_j r_j$, this is exactly the total reward minus cost.

6. *(14 points)* For a node v in a graph, the *star* of v, denoted star(v), consists of v itself and all neighbors of v in the graph. A set of nodes S form *independent stars*, if the stars of nodes $v, u \in S$ do not overlap, and are not connected by an edge. In other words, a set of nodes S forms independent stars if and only if no two nodes in S are connected by a path of length at most 3 edges. The SPANNED STARS problem is



defined as follows: The input is a graph G and a nonnegative integer k and the goal is to decide if there exists a node subset S of cardinality at least k that forms independent stars.

For example, in the network below, the node set $\{u, v\}$ does not form independent stars because star(u) and star(v) are connected by the edge between c and d. On the other hand, the node set $\{u, y\}$ does form independent stars because star(u) and star(y) are disjoint and not connected by an edge.

Prove that the SPANNED STARS problem is NP-complete.

Solution The SPANNED STARS problem is in **NP** because it has a polynomial time verifier: given a graph G, a number k, and a subset of nodes S, we can check in polynomial time that S has cardinality at least k and that S forms independent stars. We can check the latter condition by checking any two nodes in S have distance more than 3 in the graph. We can use breadth first search to compute the distance between nodes in the graph. The maximum size of the certificate S is at most n, the number of vertices, which is polynomial in the size of the instance.

Several different reductions work for this problem. The most natural starting point for the reduction is the *independent set problem*.

Reduction 1: "dangling nodes". This reduction just adds a dangling node to every node in the original graph. If S is an independent set in the graph, then the set of dangling nodes S' of S forms independent stars. (If the set S is not independent, the distance between any two nodes of S' is larger than 3.) Similarly, any set T that forms independent stars in the new graph corresponds to an independent set.

Reduction 2: "subdividing edges with super node". For this reduction we are assuming that the graph doesn't contain isolated nodes. This assumption is fine because there is an easy reduction from independent set on general graphs to independent set on graphs without isolated nodes (simply remove all isolated nodes and add them to the final independent set).

This reduction subdivides every edge by adding a new node (so that every edge in the original graph becomes a path of length 2 in the new graph). Furthermore, the reductions creates super node and connects it to every newly added node.

If S is an independent set in the original graph, then the same set forms independent stars in the new graph. Here it is important to note that super node is not part of the star of any selected node. Hence, the edges of incident to the super node cannot connect different stars.

On the other hand, if S forms independent stars in the new graph, then there exists an independent set of the same size in the original graph. Since the star of the super node overlaps with the star of any other node, we may assume that S does not contain the super node. Similarly, the star of any newly added node is connected to the star of any other node. So we can also assume that S does not contain one of the newly added nodes. It follows that S consists only of nodes of the original graph. Now the fact that the stars of S are independent in the new graph implies that the nodes S are independent in the original graph.

7. Solve Problem 2 at the end of Chapter 8 (page 505).

Solution. First we need to show that the Diverse Subset Problem in in NP. Given a set if k customers, it is easy to check if they satisfy the diversity property required.

To prove that Diverse Subset is NP-complete, we show that Diverse Subset $\geq_P$ Independent Set. Given a graph G and target size k, which is an instance of Independent Set, we will have a product corresponding to each edge, and a customer corresponding to each node, and say that a customer v has bought the products corresponding to the edges adjacent to v. We claim that graph G has an independent set of size k if and only if the constructed instance of the Diverse Subset Problem has a diverse set of customers of size k.

- Given a graph G with indecent set I, the customers corresponding to set I must be diverse, as independence implies that no two nodes in I are adjacent to the same edges.
- Given a diverse set of customers, we claim that the corresponding set of nodes I must be independent on G. This is true, as if they were not independent, i.e., we had an edge between two nodes $i, j \in I$, then the product corresponding to edge $e = (i, j)$ was purchased by both customers, contradicting the assumption that the set is diverse.

To prove that Diverse Subset is NP-complete, we show that Diverse Subset $\geq_P$ Independent Set. Given a graph G and target size k, which is an instance of Independent Set, we will have a product corresponding to each edge, and a customer corresponding to each node, and say that a customer v has bought the products corresponding to the edges adjacent to v. We claim that graph G has an independent set of size k if and only if the constructed instance of the Diverse Subset Problem has a diverse set of customers of size k.

- Given a graph G with indecent set I, the customers corresponding to set I must be diverse, as independence implies that no two nodes in I are adjacent to the same edges.
- Given a diverse set of customers, we claim that the corresponding set of nodes I must be independent on G. This is true, as if they were not independent, i.e., we had an edge between two nodes $i, j \in I$, then the product corresponding to edge $e = (i, j)$ was purchased by both customers, contradicting the assumption that the set is diverse.

8. A friend of yours is working for a political action group. The group decided to use committees to consider issues. For each decision they have to make, they consult a couple of the relevant committees. For example, the PR committee needs to be consulted on all issues that may have public relations consequences; when considering a new recruitment slogan, they would want to consult both the recruitment committee and the PR committee. The head of the organization would like to place a few of his trusted people on some of the committees to make sure that for some issue, at least one trusted person is in some committee meeting about it.

To formalize the problem the action group has n working committees $C_1, \ldots, C_n$. The issues the head cares about are $I_1, \ldots, I_m$, and each of the issues I_j will get discussed in a subset A_j of the committees. He has k trusted people. He can send a trusted person to at most one committee. His Committee Seeding problem is to decide if there are k committees that the head can choose such that each of the m issues will be discussed in at least one of them.

Prove that the Committee Seeding problem is NP-complete.

Solution. The new variant is clearly also in NP, given solution on which committees to place the k trusted member, its easy to check if all issues will be discussed by the relevant committees.

We prove that the problem is NP-complete by showing that Vertex Cover $\leq_P$ Committee Seeding. Given an input for the Vertex Cover problem, we define a Committee Seeding problem as follows. The nodes correspond to committees, and each edge (u, v) will correspond to an issue, an issue that is discussed by committees C_v and C_u. We claim that the graph has a vertex cover of size k if and only if the committee selection problem just defined with the same k is solvable.

- If S is a vertex cover in G of size k, and adding a member of each of the k committees corresponding to nodes in S makes sure all issues are discussed by a committee with a trusted member.
- Also, if S is the set of committees with trusted members in the constructed Committee Seeding problem, then S must be a vertex cover as otherwise the issue corresponding to an uncovered edge (u, v) would not be discussed by a trusted member.

Alternate Solutions. Its equally natural to use Set Cover in the reduction.

Backwards Reductions. Recall, we need to prove that Committee Seeding is hard. So for example, need to show that Committee Seeding $\geq_P$ Monotone SAT. Showing how to solve the Committee Seeding problem using Monotone SAT proves the reverse: Committee Seeding $\leq_P$ Monotone SAT, which we already know from the homework as Monotone SAT is NP-complete.