Rooms, Times, and Review for Prelim 2

- Prelim 2 is Tuesday, November 12th 7:30-9pm. We have two rooms.
- We have two rooms:
  - If your last name starts between A-Sc then your exam will be in Kennedy 116 (Call Auditorium),
  - If your last name starts between Se-Z then you need to go to Warren Hall B25.

  People taking the exam in not the assigned room, loose at least 10% of the credit.
- **If you have a conflict, or need special arrangement, you should have heard from Corey Torres (ct635) with the alternate time and room where you will take the exam.**
- We'll have an in-class **review session Monday November 11th**. And will have two review sessions by TAs:
  - Review of NP-completeness 2-3 PM (Gates G01) Saturday, November 9th by Baxter, Manish, and Yiteng
  - Review of flows/cuts 3:30-4:30 PM (Gates G01) Saturday, November 9th by Jesse and Salil

Guidelines for Prelim 2

Prelim 2 covers Flows (Chapter 7) and NP-completeness (Chapters 8). Note that only sections we covered in class are required for the prelim. Material includes classes up to Wednesday, November 6th. In this handout, we summarize what you need to know for the prelim, and then give you examples of prelim questions from previous years. Solutions for the prelim review questions will be available Sunday (November 10th).

Network Flow

The questions on network flow will test for various things. First, you were taught basic definitions concerning network flow, such as: flow network, capacity and flow conservation constraints, residual graph, augmenting path, s-t cut. You should know these definitions and their basic properties. You were taught the Ford-Fulkerson algorithm and you should know how the algorithm works, be able to run it on small graphs, and be able to answer questions about it such as its running time, or how it decides whether to terminate. You were taught the relationship between flows and cuts, including the max-flow min-cut theorem and the procedure for finding a minimum (s,t)-cut given a maximum flow in a network. You should be able to find a minimum cut in a flow network, answer questions involving the relationship between flows and cuts.

Flow reductions (i.e., reducing other problems to either max-flow or min-cut) are an important part of CS 4820, and it is a pretty safe assumption that there will be a prelim question about flow reductions. Sections 7.5, 7.6 and 7.11 are examples of this technique we covered. The hallmark of a problem that reduces to max-flow is that it involves assigning things of type A to things of type B subject to some constraints. (e.g., candidates to interviewers). The hallmark of a problem that reduces to min-cut is that it involves classifying things into two types (project selected vs. not selected).

The process of designing a flow reduction generally involves reasoning about the problem constraints and trying to figure out how they can be represented in the form of a graph. A very stereotypical graph structure consists of a source and sink, two layers of nodes that are directly connected to the source and the sink, respectively, and a bipartite graph linking the two layers of nodes together. For more advanced flow reductions, it is often necessary to make this basic graph structure a bit more elaborate, adding "gadgets" to reflect other problem constraints. Homework Problem 2 on Problem Set 6 is a good example of this. There's no "cookbook" rule for designing the gadgets in a flow reduction. Generally speaking, it's a good idea to think carefully and clearly about the problem constraints, making deductions about what types of vertices the flow network must contain and which of these types must be adjacent to each other. For example, if you're working on a problem where there are zoo animals who need care at certain times of day, and zookeepers who can only work at certain times of day, but there's no constraint on which zookeepers can care for which animals, then it's natural to guess that the graph will have three types of nodes (corresponding to animals, zookeepers, and times of day) and that the time-of-day nodes must be adjacent to the animal nodes and also adjacent to the zookeeper nodes, whereas the zookeeper nodes don't need to be adjacent to the animal nodes. On the other hand if there is a constraint on which zookeepers can care for which animals, than we probably need edges from zookeepers to animals to encode these constraints.

Proving correctness of reductions to max flow (and min cut). We note that to save time, typical flow problems on prelims only ask for construction and running time, and do not make students write out a formal proof of correctness. For feasibility problems (i.e. problems that ask you whether there exists a solution satisfying some constraints) the proof consists of proving both directions of the statement: there is a solution to the original problem if and only if there is a solution to the translated version of the problem (in this case, finding a flow of value greater than or equal to v, or a cut of capacity less than or equal to c). The first direction is generally easier. It consists of transforming an arbitrary solution to the original problem into a valid solution to the flow instance of the problem. To do this, assume that a solution to the original problem exists. Describe how to build a flow solution from it (usually by translating every "piece" of the solution into a flow path), and argue that the flow solution is valid (i.e. obeys capacity constraints, obeys flow conservation, and has the desired flow value). The second direction is generally slightly harder. It consists of transforming an arbitrary solution to the flow instance of the problem into a valid solution to the original problem. It often requires you to use the fact that if all the capacities are integers then **there is an integer flow**. To do this, assume that a solution to the flow instance exists. Use it to construct a solution to the original problem, often by interpreting the presence or absence of flow on an edge as corresponding to a yes/no decision in the original problem. Argue that the solution constructed in this way obeys all the required constraints in the original problem. Proving correctness of reductions to min-cut is similar.

For optimization problems (i.e., those which ask you to maximize or minimize some quantity, rather than just asking you to satisfy some constraints) the reasoning is similar, except that you must also show that the "quality" of a solution to the original problem (the value that you want to maximize or minimize) translates into the value of a flow or the capacity of a cut in the flow instance that your reduction creates, and conversely the value of a flow or capacity of a cut in the flow instance translates

into the "quality" of the corresponding solution to the original problem.

NP-Completeness

The other main topic on the prelim is NP-completeness. Once again, you are responsible for understanding basic definitions and concepts, such as what it means to say that there exists a polynomial-time reduction from decision problem A to decision problem B, or what it means to say that a decision problem belongs to NP, or is NP-complete.

NP-completeness proofs are an important part of CS 4820, and again it is a pretty safe assumption that there will be a problem on the prelim that asks you to show some problem is NP-complete. Suppose you encounter a question that asks you to show that a decision problem called "Foo" is NP-complete. The proof that Foo is NP-complete generally has five steps.

1. **Foo is in NP:** Show that there is a polynomial time verifier for Foo. The polynomial time verifier takes an instance of the problem Foo and a possible answer to that instance and, in polynomial time, answers 'yes' if the possible answer is a correct solution and 'no' if the possible answer is not a correct solution.

2. **Design a reduction FROM some NP-complete problem X TO Foo (showing that X $\leq_P$ Foo):** Pick an NP-complete problem (more on this below). Use an oracle that solves Foo to solve the NP-complete problem. In other words, describe how to translate an arbitrary instance of the NP-complete problem into an equivalent instance of Foo. Sometimes the translation will be straightforward and will basically consist of mapping a type of object in one problem to a type of object in the other problem. Other times the reduction will require the construction of gadgets (like in the reduction from SAT to INDEPENDENT SET). When reducing from SAT in general, there will be gadgets for binary decisions, and gadgets for enforcing the rule for clauses.

3. **The reduction takes polynomial time:** Calculate the running time of the reduction. Make sure that you aren't creating exponentially many nodes (or animals or zookeepers...).

4. **Translate arbitrary solution to NP-complete problem instance into solution to Foo instance:** This is generally the easier direction.

5. **Translate arbitrary solution to Foo instance your construction created into solution to NP-complete problem instance:** This is generally the harder direction, you need to make sure that there aren't different style solutions from the one you had in mind while doing the construction.

Don't reduce the wrong way. Remember, you want to reduce some NP-complete problem X to the target problem. If there is a way to reduce the NP-complete problem to the target problem, then the target problem is at least as difficult as the NP-complete problem, as desired. If you reduce the wrong way (i.e. design an algorithm that solves the target problem using an oracle for an NP-complete problem) then you are proving that the NP-complete problem is at least as difficult as the target problem, which is already known, as we already know X is NP-complete, and hence at least as hard as any other problem in NP.

At the risk of being repetitive, let's state this another way. When a problem says, "Prove that Foo is NP-complete," the problem is **not** asking you to give an algorithm to solve Foo. It is not even asking you to show how Foo can be solved, assuming that you already have a subroutine to solve SAT (or HAMILTONIAN

PATH, or whatever). **When you are asked to prove that Foo is NP-complete, it means you should present an algorithm for solving *some other* NP-complete problem, such as** SAT**, assuming that you *already have an algorithm that solves Foo*.**

Choosing a problem to reduce from. NP-complete problems tend to break up into a few general categories. If you can identify the category of a problem, that will tell you which NP-complete problems are most similar. These similar problems are often the easiest to reduce from. Below is a list of the general categories with a few of the examples which are most likely to be useful to you.

1. Packing problems (Independent Set)
2. Covering problems (Vertex Cover)
3. Partitioning problems with constraints (coloring)
4. Sequencing problems (Hamiltonian Cycle, or Hamiltonian Path)
5. Constraint satisfaction problems (SAT)

For a more complete list, see the Partial Taxonomy of Hard Problems in section 8.10 of the textbook.

If no problem is obviously a trivial translation, think about what gadgets you can make and what kinds of constraints those gadgets could enforce (and therefore what problems the target problem can mimic).

Some other tips on choosing what problem to reduce from.

1. If a problem asks you to decide if there exists a set of *at least k* objects satisfying some properties, try reducing from another problem that involves picking at least *k* objects, e.g. INDEPENDENT SET.
2. Similarly, if a problem asks you to decide if there exists a set of *at most k* objects satisfying some properties, try reducing from another problem that involves picking at most *k* objects, e.g. VERTEX COVER.
3. When a problem doesn't easily fit into any of the general categories listed above, the most useful starting point for reductions is the SAT problem. For some reason, it's unusually easy to design gadgets reducing SAT to other problems that don't bear any obvious relation to Boolean variables and clauses.

Acknowledgement: The parts of this document that pertain to reductions were written by Costandino Dufort Moraites for CS 4820 in a previous year.

Practice Questions for Prelim 2

The following sample questions are designed to give you a sense of the type of questions you should expect on the prelim. The questions are from previous years (*the first three were last year's prelim*, you may want to try them under prelim conditions). Solutions to the question will be discussed at the review session, and will posted on CMS after the review. You can also discuss the questions with the TAs in office hours (before or after the review session).

If you would like to try more practice problems, the first few exercises at the end of each chapter are an excellent resource for this. You can use Piazza to ask us about solutions to these problems.

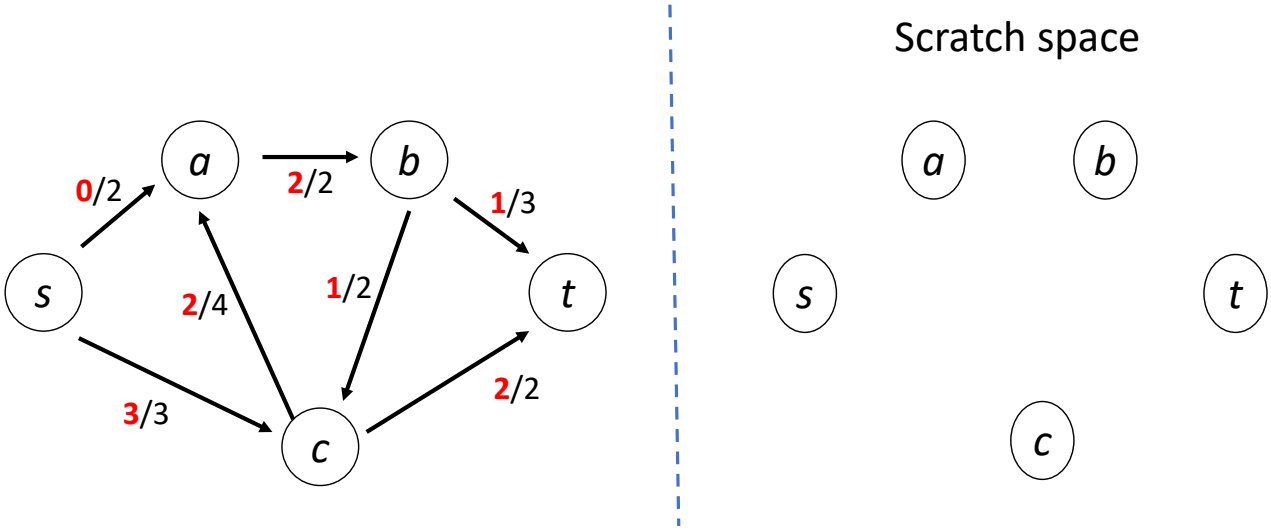
1. *(20 points)* Short answer. Each of these questions asks for a true/false answer or a short answer to a question. Follow the instructions for each question. **No proofs are required in this section.**

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.

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.)



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.

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?

(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?

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 $\sum_{i \in X} |C_i|$.

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.*

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.

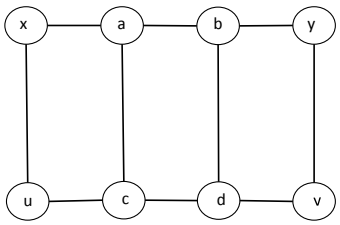
b. If students have more courses in their lists, it is easier to find a desired assignment. In order to encourage the students to do this, we tell them that when a student's list includes more than 3 courses, they can get assigned to up to 4 courses from their list.

   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.

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 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.

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 $\sum_j r_j$.

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.

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

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 commit-

tees. 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.