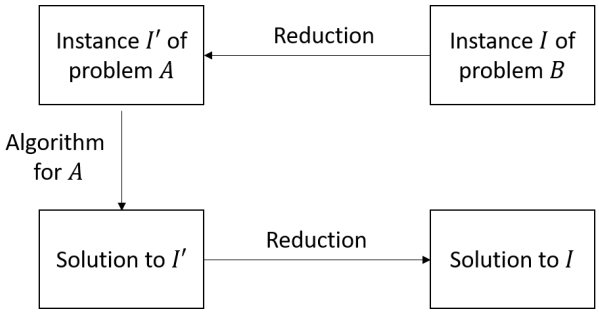
Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

Reduction: Suppose we have an algorithm to solve problem A, how can we use it to solve problem B?

This has been and will continue to be a recurring theme of the class. Examples so far include

- Use LP to solve max flow.
- Use max flow to solve min s-t cut.
- Use min s-t cut to solve global min-cut.
- Use max flow to solve max 2-flow.
- Use shortest path to solve Greatest Roads.
- Use minimum spanning tree to solve maximum spanning tree.

In each case, we would transform the instance I of problem B we want to solve into an instance I' of problem A that we can solve, and also describe how to take a solution for I' and transform it into a solution for I:



Importantly, the transformation should be efficient, i.e. takes polynomial time. If we can do this, we say that we have reduced problem B to problem A.

Conceptually, a efficient reduction means that if we can solve problem A efficiently, we can also solve problem B efficiently. On the other hand, if we think that B cannot be solved efficiently, we also think that A cannot be solved efficiently. Put simply, we think that A is "at least as hard" as B to solve.

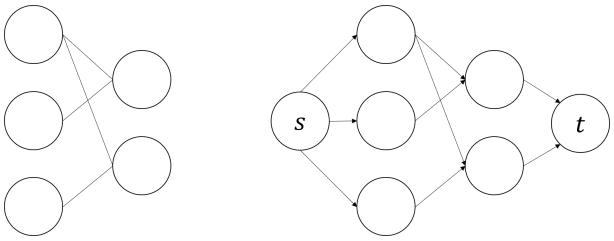
To show that the reduction works, you need to prove (1) if there is a solution for instance I' of problem A, there must be a solution to the instance I of problem B and (2) if there is a solution to instance I of B, there must be a solution to instance I' of problem A.

1 Bipartite Matching via Max Flow

Let $G = (L \cup R, E)$ be a bipartite graph such that for every edge, one of the endpoints is in L and the other is in R. Recall that a matching in G is a subset of the edges such that no two edges share an endpoint. In this problem, we will reduce finding the maximum bipartite matching to the max flow problem.

Consider the following capacitated network G' formed from G: We take G and add two new vertices s and t. All edges in G are directed from L to R. We then add a directed edge from s to all vertices in L, and a directed edge from all vertices in R to t. s is the source, and t is the sink. All edges

are assigned capacity 1. Here is one example of a bipartite matching instance and the corresponding max flow instance:



(a) Prove that for any matching in G, there is a flow of the same size in G'.

(b) Prove that the size of the max flow in G' of size is at most the size of the maximum matching G. Combined with the previous part, this implies we can find a maximum matching in a bipartite graph using an algorithm for max flow. (Hint: In any network where all capacities are 1, there is a max flow that assigns flow value 0 or 1 to every edge).

(c) We've shown that we can reduce bipartite matching to max flow. In lecture we saw an $O(|V||E|^2)$-time algorithm for max flow. Using this algorithm, what is the runtime of the bipartite matching algorithm this reduction gives us? You may assume $|E| \geq |V|$.

2 Bad Reductions

In each part we make a wrong claim about some reduction. Explain for each one why the claim is wrong.

(a) The shortest simple path problem with non-negative edge weights can be reduced to the longest simple path problem by just negating the weights of all edges. There is an efficient algorithm for the shortest simple path problem with non-negative edge weights, so there is also an efficient algorithm for the longest path problem.

(b) We have a reduction from problem B to problem A that takes an instance of B of size n, and creates a corresponding instance of A of size n^2. There is an algorithm that solves A in quadratic time. So our reduction also gives an algorithm that solves B in quadratic time.

(c) We have a reduction from problem B to problem A that takes an instance of B of size n, and creates a corresponding instance of A of size n in $O(n^2)$ time. There is an algorithm that solves A in linear time. So our reduction also gives an algorithm that solves B in linear time.

(d) Minimum vertex cover can be reduced to shortest path in the following way: Given a graph G, if the minimum vertex cover in G has size k, we can create a new graph G' where the shortest path from s to t in G' has length k. The shortest path length in G' and size of the minimum vertex cover in G are the same, so if we have an efficient algorithm for shortest path, we also have one for vertex cover.

3 A Reduction Warm-up

In the Undirected Rudrata path problem (aka the Hamiltonian Path Problem), we are given a graph G with undirected edges as input and want to determine if there exists a path in G that uses every vertex exactly once.

In the Longest Path in a DAG, we are given a DAG, and a variable k as input and want to determine if there exists a path in the DAG that is of length k or more.

Is the following reduction correct? Please justify your answer.

Undirected Rudrata Path can be reduced to Longest Path in a DAG. Given the undirected graph G, we will use DFS to find a traversal of G and assign directions to all the edges in G based on this traversal. In other words, the edges will point in the same direction they were traversed and back edges will be omitted, giving us a DAG. If the longest path in this DAG has $|V| - 1$ edges then there must be a Rudrata path in G since any simple path with $|V| - 1$ edges must visit every vertex, so if this is true, we can say there exists a rudrata path in the original graph. Since running DFS takes polynomial time ($O(|V| + |E|)$), this reduction is valid.