**COMPSCI 330: Design and Analysis of Algorithms** Spring 2019

# Lecture 24: Classical NP-hard Problems

*Lecturer: Rong Ge* *Scribe: Shweta Patwa*

**Outline:**

- Hamiltonian Path to TSP
- 3-SAT to quadratic program
- Tripartite Matching to SUBSET SUM

## 1 General recipe for doing reductions

In order to prove that B is NP-hard, given that we know A is NP-hard.
Reduce A to B (and NOT in the other direction).

Steps:

1. Given an instance $X$ of A, create an instance $Y$ of B.
2. If answer to $X$ is YES, then answer to Y is YES.
3. If answer to $X$ is NO, then answer to Y is NO; use contrapositive.

When you are trying to construct an instance Y of B, keep the second and third steps into consideration. Suppose I have a solution to X, how do I construct a solution to Y?

## 2 Hamiltonian Path to TSP

**Hamiltonian Path:** Given a graph $G$ (directed/undirected), a start vertex $s$ and an end vertex $t$, find a path from $s$ to $t$ that visits every vertex in $G$ exactly once.
Hamiltonian cycle problem is another well-known NP-hard problem.
Such a path does not always exist. A simple example is a tree where the root has three children, two of which are $s$ and $t$. Thus, this is a NO instance for the Hamiltonian Path problem.

**Travelling Salesman Problem:** Given a weighted graph $G$, and there is a salesman who wants to start at a vertex $s$, visit all vertices and come back to $s$.

Given a threshold $L$, TSP cycle problem asks whether there is a way to visit all vertices (and come back to $s$) with total edge length at most $L$.
Aside: In some sense, the starting vertex in the cycle version of this problem does not matter.

### 2.1 Reduction from Hamiltonian Path to TSP

1. Given an instance $X$ of Hamiltonian Path problem, create an instance $Y$ of TSP cycle problem.
   Similarities: visit every vertex one.
   Difference: For Hamiltonian Path problem we are given an unweighted graph, whereas for TSP we are given a weighted graph. Also, one looks at paths but the other looks at cycles.

   Use the same set of vertices. Y is a complete graph on these vertices.
   Add an edge between $s$ and $t$ in $Y$. For every edge that does not appear in $X$ but is in $Y$, assign a large weight. Set the weight of the edge between $s$ and $t$ to be 0 because we want the TSP cycle problem to be able to pick this edge. For edges in $X$ and $Y$, assign a small weight.

   $L = n-1$ in the intended solution since there are $n-1$ edges in the original graph cost.
2. If answer to $X$ is YES, then answer to $Y$ is YES.
   Select the same path in $Y$ and add the edge connecting $s$ and $t$. This gives us a valid TSP cycle in $Y$. Also, the length of the cycle is $n-1$ which equals the length of a Hamiltonian path in $X$.
3. If answer to $X$ is NO, then answer to $Y$ is NO; use contrapositive.
   If there is a cycle in the TSP problem with total length $n-1$. There are $n$ edges in the TSP cycle YES instance, of which one has a weight of 0. All other edges have weights $\geq 1$. Then the only way to have total length equal to $n-1$ is to include the edge $s-t$ and only use edges of length 1. This gives us a valid Hamiltonian Path in the original graph.

## 3 3SAT to Quadratic Programming

In quadratic programming, you can have polynomials of degree at most 2 in the given variables. An example is $x_1^2 + x_2x_3 + x_4 \leq 3$. It is very powerful, and NP-hard to solve. Under this definition, this problem is not in NP because it can involve irrational numbers and it is not clear how to represent these numbers in computers and how to verify a solution.

**Decision problem:** Given a set of quadratic constraints, does there exist a feasible solution?
Here, we are trying to use quadratic programming to solve 3SAT.

### 3.1 Reduction from 3SAT to Quadratic Programming

1. Given an instance $X$ of 3SAT, create an instance $Y$ of Quadratic Programming.

   We need gadgets for variables here so we can map solutions between the two problems.
   Variable $x_i$ is either true or false. We create variables in the quadratic program that can only take values: $0/1$. Say if $y_i$ is this variable, then $y_i^2 = y_i$ enforces that $y_i$ is either 0 or 1.
   Negation of a variable $y_i : 1 - y_i$.

   We also need gadget for clauses. Suppose we have a clause $x_1 \bigvee x_3 \bigvee \overline{x_5}$ Then, the interpretation of the associated variables will be that either $x_1$ is true, $x_3$ is true or $x_5$ is false. Thus, $y_1 + y_3 + (1 - y_5) \geq 1$.

2. If answer to $X$ is YES, then answer to Y is YES.
   If we have a satisfying assignment to the 3SAT formula, we assign 0 or 1 to $y_i$'s according to what we discussed above. This gives us a valid solution to the corresponding quadratic program.
3. If answer to $X$ is NO, then answer to Y is NO; use contrapositive.
   If we have a solution to the quadratic program, then they must correspond to $0/1$ values that can be translated to a satisfying 3SAT assignment.

# 4 Tripartite Matching to SUBSET SUM

**Tripartite matching:** Given three sets $U, V$ and $W$, each containing $n$ vertices, and hyperedges $(u, v, w)$, where $u \in U, v \in V$ and $w \in W$. A tripartite matching is a way of selecting $n$ hyperedges, so that every vertex is adjacent to a hyperedge. If we want to use $n$ hyperedges to cover all $3n$ vertices, then we must use exactly one vertex is each of the $n$ edges. Is there is a tripartite matching in the given graph?

**SUBSET SUM:** given integers $a_1, a_2, ..., a_n$ and a target $m$. Does a subset of these numbers sum up to $m$?

The dynamic program that you saw earlier is a pseudo-polynomial time algorithm because it is polynomial in the range of the values that $m$ could take, i.e., it is exponentially big in the number of bits to represent $m$ as binary digits.

## 4.1 Reduction from Tripartite Matching to SUBSET SUM

1. Given an instance $X$ of Tripartite Matching, create an instance $Y$ of SUBSET SUM.
   Tripartite Matching wants you to select some hyperedges, and SUBSET SUM wants you to pick a subset of number that sum up to a given target number.

   Thus, hyperedges in Tripartite Matching should correspond to numbers in SUBSET SUM.

   **Idea:** Use an **intermediate problem**. We use SUBSET VECTOR problem, so reduce from Tripartite Matching to SUBSET VECTOR to SUBSET SUM.

   **SUBSET VECTOR:** Given $n$ vectors $v_1, v_2, ..., v_n$ and a target vector $u$. The answer is YES iff a subset of these vectors sums up to $u$.
   This is a strictly harder problem of the target problem. It is not easy to a dynamic program to solve this.

### 4.1.1 Reduction from Tripartite Matching to SUBSET VECTORS

How do you map a hyperedge into a vector? What is the relation between a hyperedge and vertices? Here, vertices can correspond to components of vectors. As a result, we have $3n$ vertices then the dimensions of vectors will equal $3n$. Example: $(1, 1, 2)$ can get mapped to $(100|100|010)$. We want the target vector then to be $(111|111|111)$, i.e., select $n$ hyperedges so that each vertex is in exactly one hyperedge.

### 4.1.2 Reduction from SUBSET VECTORS to SUBSET SUM

Each vector needs to be mapped to a number.

Idea $(1)$: The simplest idea is to notice that in the construction, all the vectors we use are binary vectors. For these vectors, we can just think of them as binary numbers. But this is not correct because the summation of vectors is different from the sum of binary numbers. Example: $(1,0,1,0)+(0,1,1,0)=(0,1,2,0)$, but $1010+0110=1000$ in binary.

Idea $(2)$: Note that if there are no carry operations when summing numbers, then the previous idea can work. How do me ensure this? Instead of using a binary number, we use a number with a larger basis so that a carry operation cannot happen. Just need to choose base $k$ to be large enough. For $n$ $0/1$ vectors, $k > n$ is enough.

We leave proving correctness as an exercise.