CSE202
5 – NP-Complete Problems

TA: Joseph L.

I. Introduction

- Problems in computer science can generally be classified as P, NP, NP-complete, or NP-Hard. This chapter focuses primarily on the "NP-complete" class, but we will briefly describe the other categories as well.

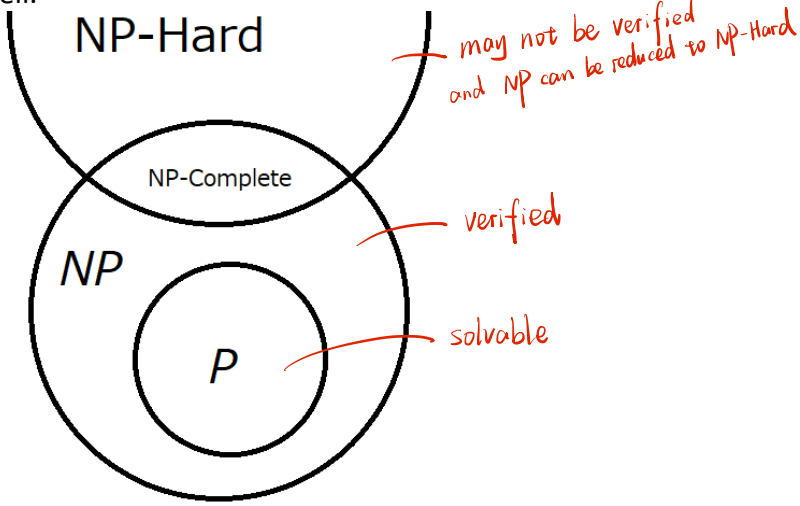

Figure 1. A Venn diagram depicting P, NP, NP-complete problems, and NP-Hard lie with respect to each other.

- The classes:
  - **P**: Problems that are *deterministically* solvable in polynomial time. This class encompasses most problems that we have discussed thus far.
  - **NP**: Problems that cannot necessarily be solved in polynomial time, but whose potential solutions can always be *verified* in polynomial time.
  - **NP-Hard**: Problems for which potential solutions may not necessarily be verifiable in polynomial time.
  - **NP-Complete**: The intersection between NP and NP-Hard.
- *All* NP-complete problems have the following characteristics:
  - The best known algorithms are *exponential* with respect to the input size.
  - As NP-Complete is part of NP, any potential solutions are verifiable in polynomial time.
  - *Any problem in NP* (including NP-complete problems) can be **reduced** to any NP-complete problem *in polynomial time*.
- A problem p is successfully "reduced" to another problem $\hat{p}$ if…
  a) Any instance of p can be converted to a corresponding instance of $\hat{p}$.
  b) Any solution s to an instance of p can be converted to a solution $\hat{s}$ of the corresponding instance of $\hat{p}$.
- If *any* NP-complete problem is found to be solvable in polynomial time, then *all* NP problems would be solvable in polynomial time (therefore, P=NP) via the following steps.
  1. Suppose we would like to solve an instance of an NP problem P, and we know a NP-complete problem $\hat{P}$ to be solvable in polynomial time.
  2. We reduce the instance of p to its corresponding instance in $\hat{p}$, which takes polynomial time.
  3. We solve the instance of $\hat{p}$, obtaining solution $\hat{s}$. This also takes polynomial time, as per our supposition.

4. We convert $\hat{s}$ to the solution s of the original problem, which takes polynomial time.
5. All of our steps took polynomial time, and thus our overall runtime is polynomial

- The "complete" in "NP-complete" signifies the fact that because any NP problem reduces to an NP-complete problem, NP-complete problems can be used to solve any problem in NP.
- Aside: Any who successfully prove (or disprove) that P=NP will be rewarded with intense admiration and gratitude from all computer scientists, as well as a paltry reward of one million dollars from the Clay Institute of Mathematics.

II. Proving NP-Completeness

Given a problem p, one need only do two things to prove p to be NP-complete: prove p to be in NP and **reduce any known NP-complete problem $\hat{p}$ to p**. It is important to note that the *opposite*, reducing p to a known NP-complete problem, proves nothing since by definition, any problem in NP can be reduced to an NP-complete problem.

Generally, we should approach these proofs with the following steps:

1. Prove p to be in NP by showing that any candidate solution can be verified in polynomial time.
2. Find a suitable problem to reduce. For example, if we're dealing with a graph problem, then we might want to try a graph-based NP-complete problem like independent sets.
3. Figure out a connection between $\hat{p}$ and p. If you can't think of a concrete connection, consider returning to step 1.
4. Do the reduction.
5. Prove that your reduction works: an instance of p has a solution if and only if the corresponding instance of $\hat{p}$ has one, and show the relationship between the solutions.
6. Prove that your reduction works in polynomial time with respect to the size of the input.

We will end this handout on two examples; these assume that you've already read and/or learned in lecture about the "independent set" and "3-SAT' problems.

III. Example 1: Clique

In a graph G, a clique is a group of nodes in which every member has an edge to every other member.

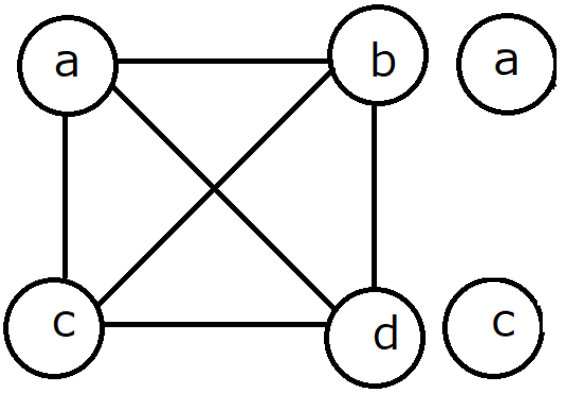

Figure 2. A clique of size 4.

Figure 3. An independent set of size 4 in the complement graph.

We will prove the following problem to be NP-complete:

Given a graph G, determine whether it has a clique of size at least k.

1. Given a group of k nodes, we can check in polynomial time ($O(k^2)$ to be precise) whether those k nodes form a clique – simply check whether all required edges are present. Therefore, this problem is in NP.

2. This is a graph problem, and the presence of edges is a key factor. This seems related to the independent set problem.
3. Recall the definition of an independent set: it is a set of nodes in which none of the members have edges to each other. On the contrary, in a clique, *all* members have edges to each other. Therefore, if a group of nodes s forms a clique in a graph G, then the same group of nodes forms an independent set in the *complement* graph, $\bar{G}$.
4. Our reduction: we start with the problem of finding an independent set of size at least k in a graph $\bar{G}$. From (2), we see that this is equivalent to finding a clique of size at least k in the complement G. Therefore, we have reduced the independent set problem to the clique problem.
5. In an independent set of size $\geq k$, no nodes have edges to each other. In the complement graph, all of these nodes have edges to each other, forming a clique. Thus, the clique problem has a solution(s) if and only if the corresponding independent set problem, on the complement graph, does.
6. Suppose $\bar{G}$ (for our independent set problem) has $|V|$ nodes and $|E|$ edges. There can be at most $O(|V|^2)$ edges ($|V|$ choose 2), so we will substitute $|V|^2$ for $|E|$.
   a. Building the complement graph G would then take $O(|V|^2 + |V|) = O(|V|^2)$. We now have the graph on which our clique problem is done, so we've reduced the problem in polynomial time.
   b. Converting a solution of one problem to the corresponding solution of the other problem is $O(1)$ since in this case, it's the exact same set of nodes. Thus, converting between solutions is also polynomial.

We have now successfully reduced the independent set problem to the clique problem, thus proving the latter to be NP-complete.

IV. Examble 2: 4-SAT

Consider another satisfiability problem in which each clause has up to 4 variables.

$$(x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge \ldots$$

We already know 3-SAT to be NP-complete and this suggests that 4-SAT should be at least as hard, if not harder. We'll reveal their relationship through a formal reduction.

1. Given a set of variable assignments for a 4-SAT problem with k clauses, we can check whether it's a solution in $O(k)$. Therefore, 4-SAT is in NP.
2. As this is another satisfiability problem, 3-SAT would seem the best choice for a reduction. Thus, we will reduce 3-SAT to 4-SAT.
3. The following question is key: is there a way to represent a clause of 4 variables using several clauses of 3 variables each? With some thought, we arrive at the following equality:

   $$(x_1 \vee x_2 \vee x_3 \vee x_4) = (x_1 \vee x_2 \vee y_1)(\neg y_1 \vee x_3 \vee x_4)$$

   On the left-hand side, we need at least one of x_1, x_2, x_3, x_4 to be true.
   In the right-hand side, we first need either of x_1, x_2 to be true, or otherwise for a new variable y_1 to be true. If y_1 is true, then we can still satisfy the circuit even if x_1, x_2 are both false – in that case, however, the second clause tells us that one of x_3, x_4 must be true (since y_1 and $\neg y_1$ can't both be true).
   Thus, we see that the left- and right-hand sides are equivalent, and this gives us a way to convert between 3-SAT and 4-SAT.
4. Our reduction: start with a 3-SAT instance in which each clause can be paired with another such that one variable in the first clause reappears negated in the second clause.

   $$(x_1 \vee x_2 \vee y_1)(\neg y_1 \vee x_3 \vee x_4)$$

   As we saw earlier, such a pair of clauses can be represented with one clause with 4 variables.

   $$(x_1 \vee x_2 \vee x_3 \vee x_4)$$

Thus, we can reduce these 3-SAT instances to their corresponding 4-SAT instances by doing this conversion for each pair of clauses.

5. In our conversion, each pair of 3-SAT clauses is satisfied by exactly the same variable assignments as the corresponding 4-SAT clause. Thus, our 4-SAT problem has a solution if and only if the corresponding 3-SAT problem has a solution.
6. Suppose our 3-SAT problem has k clauses.
   a. Each pair of clauses can be converted to a corresponding 4-SAT clause in $O(1)$ since the number of variables per clause is constant. If we do this for k clauses, then our total complexity is $O(k)$ and is thus polynomial.
   b. We can convert between solutions in $O(1)$ since the same set of variable assignments satisfies both the 3-SAT problem and the 4-SAT problem. Thus, our conversion between solutions is also polynomial.

We have successfully reduced 3-SAT to 4-SAT, and have thus proven 4-SAT to be NP-complete. We see that, perhaps surprisingly, 4-SAT isn't necessarily more complicated than 3-SAT since they're both NP-complete. As it turns out, any boolean satisfiability problem N-SAT is NP-complete for $N \geq 3$.

References

1. Kleinberg, Jon, and Éva Tardos. *Algorithm Design*. Boston: Pearson/Addison-Wesley, 2006. Print.