# CS-600-A Homework 10

Zijing Huang 10414952

**R-17.3 Show that the problem SAT, which takes an arbitrary Boolean formula S as input and asks whether S is satisfiable, is *NP*-complete.**

If we need to prove any problem is NP-complete, then we need to prove two parts as follows:

1. This problem belongs to NP.
2. This problem is NP-hard, which we can prove by finding an exist NP-hard problem and reduce it to this problem in a Polynomial time.

To prove 1 in this case, we can assign a set of arbitrary Boolean variable to verify if S is satisfiable in a polynomial time. So SAT belongs to NP. For 2, we know that CNF-SAT is kind of SAT which means we can reduce SAT to CNF-SAT. And according to Theorem 17.7: CNF-SAT is NP complete (page 490 on the textbook), So SAT is NP-hard and NP-complete.

**R-17.7 Show that the CLIQUE problem is in *NP*.**

As the proof of Lemma 17.2 in the textbook, we construct a nondeterministic algorithm for accepting instance of CLIQUE. We use the choose method to “guess” whether there is an edge between every pair of distinct vertices in C in the specific set of vertices of size k. And this guess can be computed in a polynomial time. So CLIQUE problem is in NP.

**C-17.10 Define INDEPENDENT-SET as the problem that takes a graph G and an integer k and asks whether G contains an independent set of vertices of size k. That is, G contains a set I of vertices of size k such that, for any v and w in I, there is no edge (v, w) in G. Show that INDEPENDENT-SET is *NP*-complete.**

As R-17.3 above, we need to prove the following things:

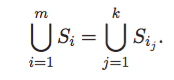
1. INDEPENDENT-SET belongs to NP.

2. INDEPENDENT-SET is NP-hard, which we can prove by finding an exist NP-hard problem and reduce it to this problem in a Polynomial time.

To prove 1, we can guess a specific subset of G with k vertices and verify whether there are two adjacent vertices, which run in polynomial time (just scan the graph structure once). So it is in NP.

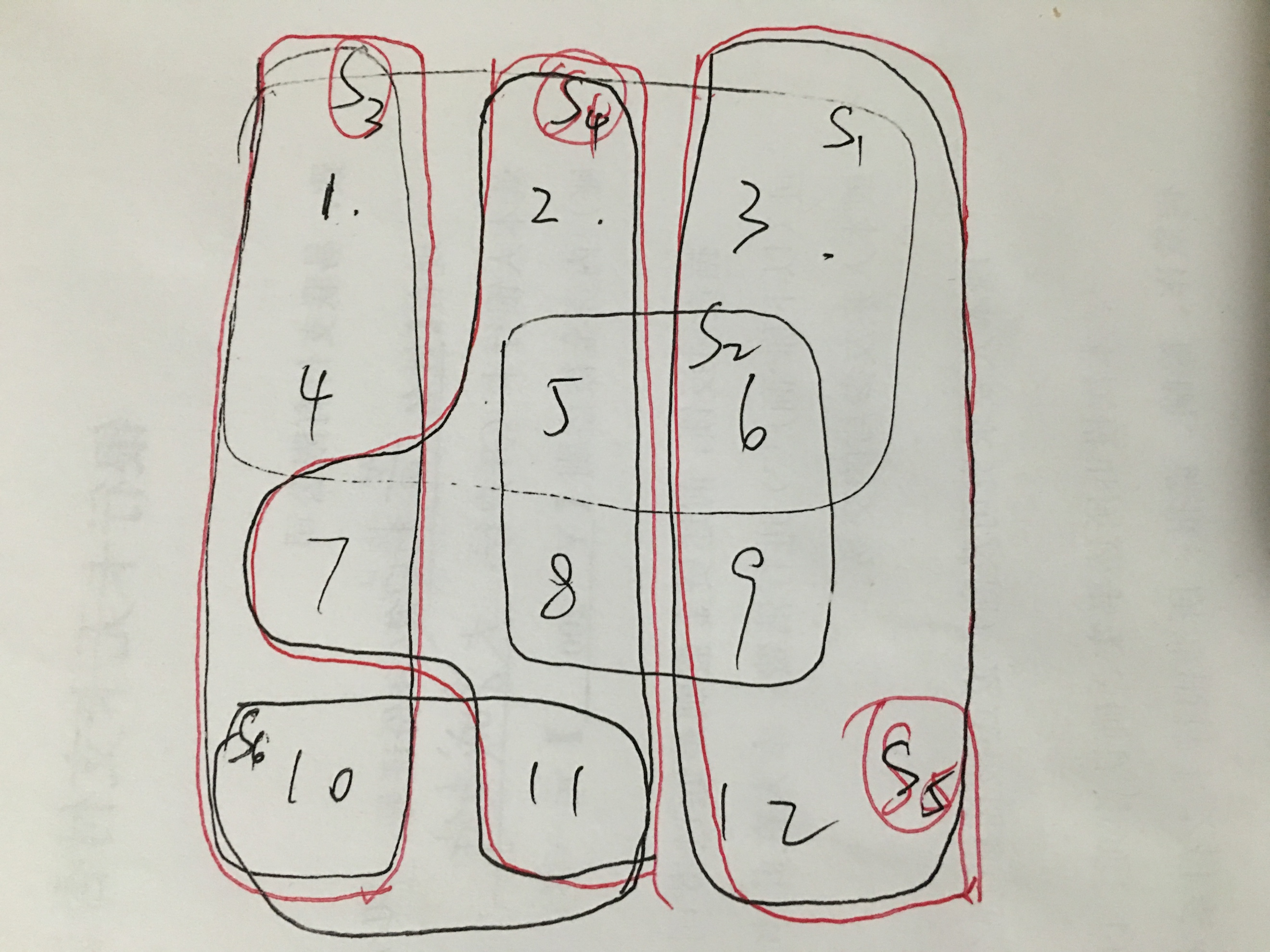
For 2, if we want to know whether there is vertex cover of size k in G, we can just check if there is an independent set of size n-k. If there is independent set, then G doesn’t have vertex cover of size k in G. That is to say, we can reduce VERTEX COVER problem to independent-set problem. And because we know VERTEX COVER problem is NP-hard (Theorem 17.9), we get INDEPENDENT-SET is NP-hard and NP-complete.

**A-17.5 Suppose you are computer security expert working for a major company, CableClock, any you have just discovered that many of the computers at CableClock are infected with malware that must have come from users visiting unsafe websites. For each infected computer, you are given a log file that lists all websites it has visited since the last time it was scanned for malware. Unfortunately, as you look over these log files, you notice that there isn’t a single website that they all visited. You conclude, therefore, that there must be a number of websites that are able to inject this malware, and the most likely candidates would be in a smallest collection that is visited by all the infected computers. Show that the decision version of the problem of determining such a collection is *NP*-complete.**

The question is asking us to find the smallest collection of websites that is visited by all the infected computers. That is suppose we have 2 infected computers Ca and Cb, and two websites W1 and W2. If W1 is visited by Ca, W2 is visited by Cb. Then the smallest collection would be {W1, W2}. If W1 is already visited by Ca and Cb, then the smallest collection could be only {W1}. So this is a SET-COVER problem. Think that S is a specific website being visited by which infected computer. For example, S1 represent W1 being visited by infected computer C1, C2, C4. So S1 = {C1, C2, C4}. And we will find an integer k, such that  while the left of the equation is all the infected computer. And we should compute a smallest k amount all the k that satisfy the equation above. We already known the SET-COVER problem is np-complete (Theorem 17.11). And SET-COVER problem could reduce to this problem. So this problem is np-complete.

**R-18.11 Suppose we are given the following collection of sets: S1 = {1,2,3,4,5,6}, S2 = {5,6,8,9}, S3 = {1,4,7,10},  S4 = {2,5,7,8,11}, S5 = {3,6,9,12}, S6 = {10,11}. What is the optimal solution to this instance of the SET-COVER problem and what is the solution produced by the greedy algorithm?**

For this instance, we can quickly get the result through a Venn Diagram as follow:



Since the set is limited here, we can easily find the solution {S3,S4,S5} which have high line by the red pen above.

And I would like to simulate the process of algorithm 18.7 in this case. Suppose R represent the rest of element.

Round 1:

R={1,2,3,4,5,6,7,8,9,10,11,12}

Always choose the a set S that has the maximum number of uncovered elements, so we choose S1 in this round.

C={S1}

Round 2:

R={7,8,9,10,11,12}

S4 has the maximum number of uncovered elements

C={S1,S4}

Round 3:

R={9,10,12}

S5 has the maximum number of uncovered elements

C={S1,S4,S5}

Round 4:

R={10}

S3 contains the last value

C={S1,S4,S5,S3}

R={} end the loop

So, the solution of greedy is {S1,S4,S5,S3}.

**C-18.1 Consider the general optimization version of the TSP problem, where the underlying graph need not satisfy the triangle inequality. Show that, for any fixed value δ ≥ 1, there is no polynomial-time δ-approximation algorithm for the general TSP problem unless *P* = *NP*.**

***Hint:* Reduce HAMILTONIAN-CYCLE to this problem by defining a cost function for a complete graph H for the n-vertex input graph G so that edges of H also in G have cost 1 but edges of H not in G have cost δn more than 1.**

As the Theorem 18.1(on page 512) says, the 2-approximation algorithm for the METRIC-TSP optimization problem depends heavily on the fact that the cost function on the graph G satisfies the triangle inequality. Now we consider the situation that not satisfy the triangle inequality.

Suppose that given an NP-hard problem X, we can produce in polynomial time a minimization prob- lem Y such that “yes” instances of X correspond to instances of Y with value at most k (for some k), but that “no” instances of X correspond to instances of Y with value greater than k. Then, we have shown that, unless P D NP, there is no polynomial-time -approximation algorithm for problem Y . (references: *Introduce to Algorithem Theorem 35.3*)

According to the hint above, we reduce HAMILTONIAN-CYCLE to this problem by defining a cost function for a complete graph H for the n-vertex input graph G so that edges of H also in G have cost 1 but edges of H not in G have cost δn more than 1 \* (1+ δn). That is to say, if you want to include one of the edges of H not in G, the total cost would be n-1+(1+ δn) = n + δn = n \* (1 + δ). But if you just include the edges of H in G, it at most cost n. So, if we want an exact polynomial-time δ-approximation algorithm for the general TSP, then we can only do it by using only the edges of H also in G to build the cycle.

**A-18.3 Suppose you work for a major package shipping company, FedUP, and it is your job to ship a set of n boxes from Rhode Island to California using a given collection of trucks. You know that these trucks will be weighed at various points along this route and FedUP will have to pay a penalty if any of these trucks are overweight. Thus, you would like to minimize the weight of the most heavily loaded truck. Assuming you know the integer weight of each of the n boxes, describe a simple greedy algorithm for assigning boxes to trucks and show that this algorithm has an approximation ratio of at most 2 for the problem of minimizing the weight of the most heavily loaded truck.**

We can define a greedy algorithm like when there is a new box, we always throw it to the least loaded truck. If there are more than one “the least loaded truck”, randomly pick one of them.

Suppose we at least have 2 trucks and 2 boxes. Traditionally, we put 2 boxes in the first truck, Tt1 = 2, Tt2 = 0. But if we use greedy method Tg1 =1, Tg2 = 1. Tt1/Tg1=2. And along with the number of truck or number of box getting larger, this coefficient getting larger too. So this algorithm has an approximation ratio of at most 2 for the problem of minimizing the weight of the most heavily loaded truck