4810-1183: Approximation and Online Algorithms with Applications
Lecture Note 2: NP-Hardness

Introduction

Suppose that you submitted a research manuscript to a journal. It is very unlikely that your paper will be accepted at the first review. Reviewers usually ask you to do more tasks, which could make your work look more completed. They might ask you to solve a particular problem.

You did try to solve the problem, but it seems to be very hard. The deadline for the resubmission is getting close. You want to tell the reviewers that "it is not possible to solve the problem", but they might not believe you and reject your paper. In this lecture, we will discuss how to "formally" say that a solution for the problem is not possible.

Actually, there is no formal way to say that. Instead, we can try to say that "many great researchers have already tried to solve the problem but failed". We can hire 100 great researchers, have them solve the problems, and claim that the problem is hard when it is not solved. However, that could be too costly for just a publication.

However, there are a hard problem that "many great researchers have already tried but failed". Most computer scientists believe that the hard problem cannot be solved. Instead of proving that the problem cannot be solved, we will prove that "if we cannot solve the hard problem, we cannot solve the reviewers' problem". Your problem cannot be solved based on the assumption that the hard problem cannot be solved. However, the assumption is very widely believed. Everyone including the reviewers should believe in that assumption. Because of that, by proving the statement, they should believe that their problem is also hard. By contrapositive, the proposition in the previous sentence is equivalent to "If we can solve the reviewers' problem, we can solve the hard problem".

Reduction

Let us discuss how to prove the proposition "If we can solve the reviewers' problem, we can solve the hard problem".

We will assume that "we can solve the reviewers' problem". Because we can solve it, there is a program for the reviewers' problem that can terminates in polynomial time. Suppose that the program is written in a library you can download in the internet, and we can call the library by the following way:

```
Output ReviewerProblem(Input1, Input2, …, InputK);
```

Based on the assumption, we will prove that "we can solve the hard problem". We will write a program that can solve the hard problem. Inside the program, we can use the function ReviewerProblem as many times we want, as far as the running time of the function `HardProblem` is still polynomial.

```
Output HardProblem(Input1, Input2, …, InputM){
      //Your code here
}
```

If you can successfully write the program, then you can prove the proposition. We can formally state that the reviewers' problem is not likely to be solved. It is interesting that, when you want to show that a problem is solvable, you have to write a program for the problem. When you want to show that a problem is not solvable, you also have to write a program for the other problem.

When we can prove that "If we can solve the reviewers' problem, we can solve the hard problem". We can informally say that "the reviewers' problem is not easier than the hard problem".

The Hard Problem

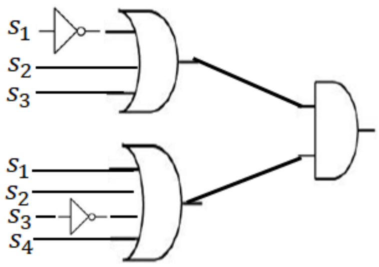
Now, it is the time to define what the hard problem is. The problem is called as *satisfiability*, which can be informally described as follows:

Input: A logic circuit that has two levels. The first level is for *or* gate, and the second is for *and* gate. One input can enter more than one different *or* gates. There might be *not* gates in front of the *or* gate.

Output: Yes or No

Constraint: Yes, if it is possible to make the circuit output "true". No, otherwise.

An input example is shown in the below image. We can have the output "true" when s_1 is true, s_2 is true, s_3 is false, and s_4 is true. Thus, when the input is the below logic circuit, the output of satisfiability problem should be yes.



Because many great researchers have tried to solve satisfiability problem but failed, everyone believe that it is a good benchmark. We call a problem that is not easier than satisfiability an *NP-hard problem*.

Now, let consider the following problem.

Input: A logic circuit that has two levels. The first level is for *or* gate, and the second is for *and* gate. One input can enter more than one different *or* gates. There might be *not* gates in front of the *or* gate.

Output: Values (true or false) for all inputs to the logic circuit.

Constraint: If it is possible to make the circuit output "true", output the values that make the circuit true. Otherwise, output an arbitrary value.

We will prove that the problem is NP-Hard. To do that, we will assume that there exists a library for our problem.

```
Values ourProblem(circuit c);
```

Then, we can write the following program for satisfiability.

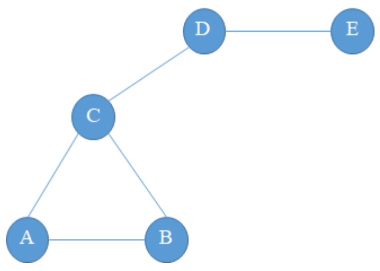
```
boolean satisfiability(circuit c){
        Values v = ourProblem(c);
        If the circuit c outputs true from the values v:
                return yes
        Else return no
}
```

When the circuit c outputs "true" from the values v, we know that it is possible to have "true" from the circuit. The output of `satisfiability` should be yes. On the other hand, the function `ourProblem` will try its best to find values that lead to true output at the logic circuit. The only case that v does not lead to the true output is when it does not exist. Because of that, the output of `satisfiability` should be no in that case. We know that the above `satisfiability` function can be used to the satisfiability problem.

We can conclude that our problem is not easier than satisfiability, and is NP-Hard problem by the program.

Hardness of Densest Subgraph Problem

In this section, we will study a problem in social networks. A social network consist of a set of persons, and a set of friendships between two persons. For example, a network might contain 5 persons, $\{A, B, C, D, E\}$, and a set of friendship, $\{\{A, B\}, \{B, C\}, \{A, C\}, \{C, D\}, \{D, E\}\}$. We know that A is a friend of B and C, B is a friend of A and C, C is a friend of A, B, and D, D is a friend of C and E, then E is a friend of D.



We want to find a set of celebrities in a social networks. There are many ways to do that, but we observe that celebrities usually know a lot of other celebrities through a number of events and parties organized every months. A number of relationships between celebrities should be large. Because we want to find a set of celebrities, we set the following optimization model:

Input:	A social networks, expected number of celebrities
Output:	A set of persons that might be celebrities
Objective Function:	Maximize the number of friendship between persons in the selected set

The mathematical formulation of the previous problem is as follows:

Input:	Set V (persons), set $E \subseteq \{\{u, v\}: u, v \in V\}$ (friendships),
	Positive integer k (number of celebrities)
Output:	$S \subseteq V$
Objective Function:	Maximize $|\{e \in E: e \subseteq S\}|$

We call the problem "densest subgraph problem". We want to show that the problem is NP-hard, so we have to write a program for satisfiability based on a library for the problem.

There is a very clear relationship between our problem and satisfiability in the previous section. We do not have that clear relationship for the "densest subgraph problem". A library for the densest subgraph problem might not be very helpful for satisfiability. Luckily, a very large number of problems are proved to be NP-hard. Those problems are known to be not easier than satisfiability. If we can prove that our problem is not easier than them, then we know that our problem is not easier than satisfiability. A collection of NP-hard problems can be found in [1]. We can search for the most similar problem and begin the proof there.

Among problems in the collection, let us choose k-clique problem, which can be formally defined as follows:

Input: Set V, set $E \subseteq \{\{u, v\}: u, v \in V\}$, integer k

Output: Yes *or* No

Constraint: Yes, if there is S such that $|S| = k$ and, for all $v_1, v_2 \in S$, $\{v_1, v_2\} \in E$. No, Otherwise.

In our social network settings, the output of the k-clique problem is yes, if there exists S such that every members of S are friends of all other members.

The k-clique problem is proved to be not easier than satisfiability. We will prove that the densest subgraph problem is not easier than k-clique problem. That will immediately imply that the densest subgraph problem is not easier than the satisfiability problem.

To prove that the densest subgraph problem is not easier than k-clique problem, we will write a program for k-clique problem based on the assumption that a library for the densest subgraph problem is provided. The program is as follows:

```
Set densestSubgraph(Set V, Set E, int k);

boolean kClique(Set V, Set E, int k){
        Set S = densestSubgraph(V, E, k)
        If, for all v1, v2 in S, {v1, v2} is in E:
                return true
        Else
                return false
}
```

When every members of S from `densestSubgraph` are friends of all other members, then we know that there exists such a set. The output of the k-clique problem should be yes. On the other hand, the output from `densestSubgraph` is supposed to be a group of persons with maximum number of friendships. If, even in a group with maximum number of friendships, we miss some friendships, we should miss some friendships in every groups. The output of the k-clique problem should be no. The program we write in `kClique` corresponding to that idea, so it is a valid program for k-clique problem.

By the previous paragraphs, `densestSubgraph` is not easier than k-clique, and the problem is NP-hard.

NP-Completeness

You might have not heard about NP-hardness but NP-completeness. In my opinion, NP-hard has more practical meanings for optimization models, but we would like to define NP-completeness for the self-containment of this manuscript.

While an NP-hard problem is a problem that is not easier than satisfiability, an NP-complete problem is a problem that is "as hard as" satisfiability. To prove that a problem is "as hard as" the satisfiability problem, the proof has to be done in 2 ways. We have to show that our problem is not easier than satisfiability, and we have to show that satisfiability is not easier than our problem. To get that, we need a program for satisfiability based on a library for our problem, then a program for our problem based on a library for satisfiability.

A collection of NP-complete problems can also found in [1]. Those problems are all as hard as satisfiability. In the previous paragraph, we can replace satisfiability with any of the problems, when we want to prove that a particular problem is NP-complete.

It is very interesting that a very large number of IST optimization models have the same hardness.

Product Selection Problem

In this section, we will give you an example how NP-hard concepts are used in recent research. The first recent problem that will be introduced is called "product selection problem" [2].

Suppose that we are a notebook company. We want to issue new models and our R&D section propose a number of possible models. We have a budget to issue only a smaller set of possible models, and we want to choose a set that optimize our profit.

Again, you have to design the best optimization model for this setting. There are many ways to formalize the problem here, and you have to think how to make the models similar to practice. Let us believe that customers choose a computer based on only 3 factors, weight, speed, and harddisk space. They will have their own specification, and they will choose a computer that satisfy their specifications with lowest price.

For example, suppose that our R&D section propose 3 models with the following information:

- Model A: Weight 2 kg, Speed 1.5 MHz, Space 1 TB, Price 70,000 Yen
- Model B: Weight 1.3 kg, Speed 1.5 MHz, Space 500 GB, Price 100,000 Yen
- Model C: Weight 1.3 kg, Speed 1.8 MHz, Space 1 TB, Price 170,000 Yen

Our competitive have the following notebook model in the market.

- Model D: Weight 1 kg, Speed 1.8 MHz, Space 2 TB, Price 300,000 Yen
- Model E: Weight 2.5 kg, Speed 1.5 MHz, Space 500 GB, Price 50,000 Yen
- Model F: Weight 0.9 kg, Speed 0.7 MHz, Space 200 GB, Price 100,000 Yen

We predict that there are 4 customers in the market with the following specifications:

- Customer α: Weight ≤ 1.4 kg, Speed ≥ 1.6 MHz, Space ≥ 1.5 TB
- Customer β: Weight ≤ 3 kg, Speed ≥ 1 MHz, Space ≥ 1 TB
- Customer γ: Weight ≤ 2 kg, Speed ≥ 1.5 MHz, Space ≥ 1 TB
- Customer η: Weight ≤ 2 kg, Speed ≥ 1 MHz, Space ≥ 500 GB

Suppose that, out of the 3 models proposed by R&D section, we can produce only 2 models. Let us consider the case that we should model A and B. Customer α will choose Model D as it is the only model available for him. Customer β can choose among model A, C and D, and he will choose model A. Customer γ can choose among model A, C and D, and he will choose model A. Customer η can among model A, B, C, D, and he will choose model A. We will have 3 customers. Actually, this choice does maximize the number of customers, so we believe that the choice is the most desirable.

It is now time to specify the problem using a mathematical model. The model below is more generalized than discussed above. We do not limit to only weight, speed, and space, but anything can be the factors. Also, we can have more than 3 factors.

<u>Input</u>:	Number of proposed models – positive integer n
	Properties of each proposed models – vectors of positive real numbers $P_1, \ldots, P_n$
	Price for each proposed models – positive integers $p_1, \ldots, p_n$
	Number of models from competitors – non-negative integer m
	Properties of models from competitors – vectors of positive real numbers $Q_1, \ldots, Q_m$
	Price of models from competitors – positive integers $q_1, \ldots, q_m$

Number of customers – positive integer d
Specification of each customers – vectors of positive real numbers $C_1, \dots, C_d$

Number of models going to production – positive integer k

Output: Set of models going to production $S \subseteq \{1, \dots, n\}$

Constraint: $|S| = k$

Objective Function: Our price to customer i, $f_i = \min_{j \in S: P_j \leq C_i} p_j$

Competitors' price to customer i, $f_i' = \min_{j: Q_j \leq C_i} q_j$

Maximize $|\{i \in \{1, \dots, d\}: f_i < f_i'\}|$

We call the above problem as the produce selection problem. To prove that the problem is NP-hard, we can search for the problem in [1]. However, for this case, it is much easier when we use recent research results in [3]. If we read a lot of research papers or listen to a lot of research talk, you will turn to know more NP-hard problems. That will help when you want to prove that a particular problem is NP-hard.

The problem in [3], called k-most representative skyline operator is defined as follows:

Input: positive integer n, vectors of positive real numbers $P_1, \dots, P_n$
positive integer d, vectors of positive real numbers $C_1, \dots, C_d$
positive integer k

Output: $S \subseteq \{1, \dots, n\}$

Constraint: $|S| = k$

Objective Function: Maximize $\left|\{i \in \{1, \dots, d\}: \text{there exists } j \in S \text{ such that } C_i \leq P_j\}\right|$

Let us get back to the product selection problem. Consider the case that there is no model from competitors. In that case, $m = 0$ and $f_i' = \min_{j: Q_j \leq C_i} q_j$ is not defined as it is the minimum of empty set. We can informally consider the non-defined value as infinity, so $f_i' \to \infty$ for all i. On the other hand, $f_i \to \infty$ only when there is no $j \in S$ such that $P_j \leq C_i$. Because of that, $f_i < f_i'$ if and only if there is $j \in S$ such that $P_j \leq C_i$. We know that, when $m = 0$,

$$|\{i \in \{1, \dots, d\}: f_i < f_i'\}| = \left|\{i \in \{1, \dots, d\}: \text{there exists } j \in S \text{ such that } C_i \leq P_j\}\right|,$$

and the two problems are equivalent.

Because of the discussion in the previous paragraph, when we have a library for the product selection problem, we can write the following program for the k-most representative skyline operator problem:

```
Set productSelection(int n, Vector[] P, double[] p, int m, Vector[] Q, double[] q,
                                                      int d, Vector[] C, int k);

Set kMostRepresentativeSkyline(int n, Vector[] P, int d, Vector[] C, int k){
       Let p be any arbitrary array length n;
       return productSelection(n, P, p, 0, [], [], d, C, k);
}
```

Thus, the product selection problem is NP-hard problem.

Exercises

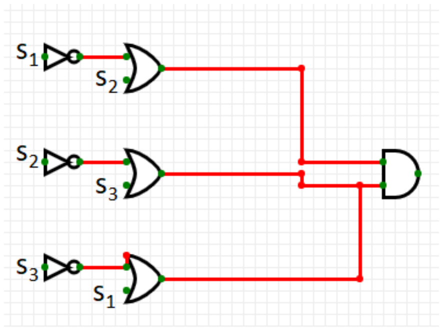
Recall the satisfiability problem. We have two layers, the *or* layer and the *and* layer there. The circuit will be satisfied if all the outputs from the *or* layer is *true*. Let us suppose that we do not want to have all the outputs from the first layer be *true*, but have the number of *true* outputs as large as possible.

Question 1: What is the input of your optimization model?

Question 2: What is the output of your optimization model?

Question 3: What is the constraint of your optimization model?

Question 4: What is the objective function of your optimization model?



Question 5: What would be a desired output of your optimization model when the input is the above circuit?

Question 6: Show that your optimization model is NP-Hard.

Instead of working on *true* and *false* value, we will have *true* = 1 and *false* = 0 as in many computer languages.

Question 7: Discuss why the most upper *or* gate outputs *true* if and only if $(1 - s_1) + s_2 \geq 1$.

Question 8: Discuss why this circuit can be satisfied if there are $s_1, s_2, s_3 \in \{0,1\}$ such that

- $(1 - s_1) + s_2 \geq 1$,
- $(1 - s_2) + s_3 \geq 1$, and
- $(1 - s_3) + s_1 \geq 1$.

Question 9: Consider the following input to CPLEX. What is the reason between the input and the satisfiability conditions discussed Question 8?

```
Minimize
s1
st
-s1 + s2 >= 0
-s2 + s3 >= 0
-s3 + s1 >= 0
end
```

Question 9: Check which outputs you have got from CPLEX. Can you say anything about the output of satisfiability based on the outputs?

Question 10: Although we can find a desirable output for this input, discuss why we cannot use CPLEX to find a desirable output for any particular inputs.

References

[1] M. R. Garey, D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, 1983.

[2] S. Xu, J. C. Lui, “*Product selection problem: Improve market share by learning consumer behavior*”, Proc. of KDD’14, pp. 851-860, 2014.

[3] X. Lin, Y. Yuan, Q. Zhang, Y, Zhang, “*Selecting stars: The k most representative skyline operator*”, Proc. of ICDE’07, pp. 86-95, 2007.