**4810-1183: Approximation and Online Algorithms with Applications**
**Lecture Note 1: Course Overview, Optimization Models, Linear Programming**

**What do we aim at this class?**

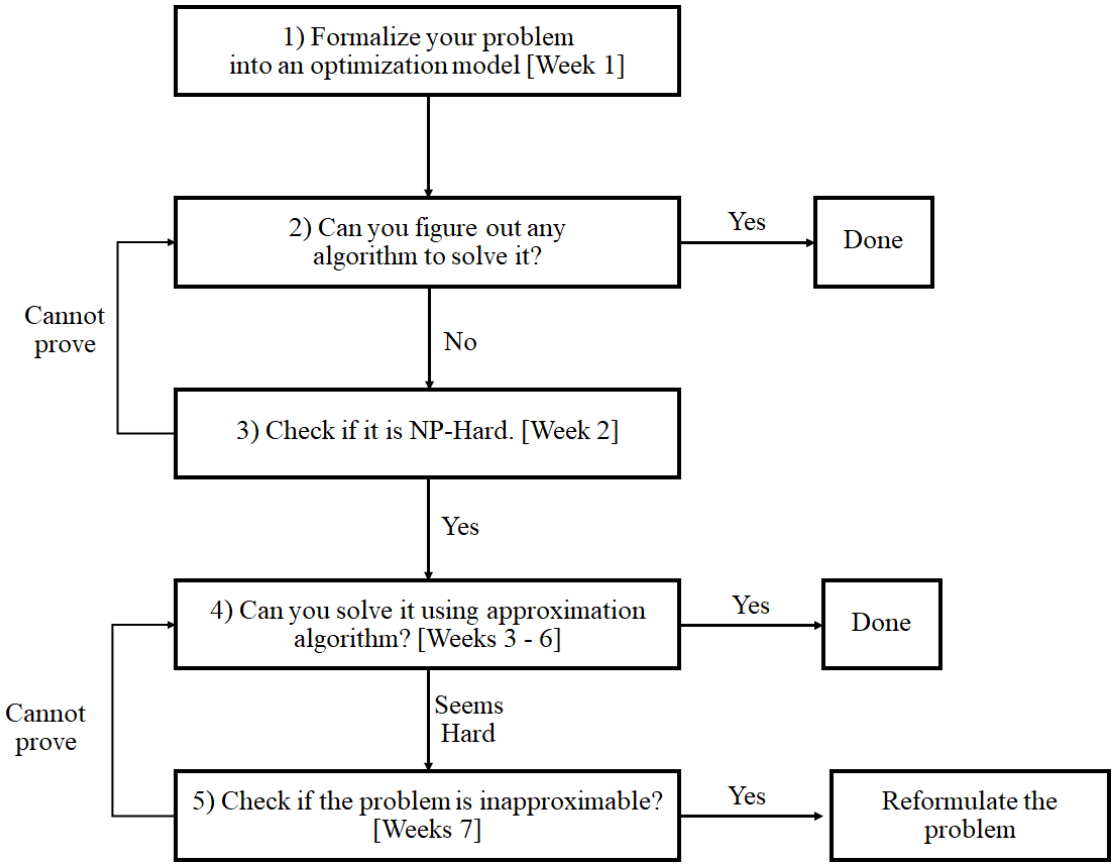
During many undergraduate courses in CS, students have a chance to join several "application-oriented courses" such as communication networks, databases, OS, or machine learnings. On the other hand, students have a chance to learn several theoretical CS concepts such as computational complexity, hardness, and data structures. However, for some of the students, the "application-oriented courses" and the theoretical CS concepts might look totally separated as they are in totally different worlds.

At the graduate level, we expect students to produce cutting-edge research results by merging everything they have learnt. However, the merging is not very easy as things are taught separately.

Because of that, in this class, I will guide you to the beginning of the merging. I will raise a lot of problems in the application-oriented courses, and will show you how to solve it using theoretical CS concepts. As some of you have not learnt enough theoretical CS concepts during the undergraduate courses, I will try to give some more theoretical backgrounds, in particular on the hardness, approximation algorithms, and online algorithms. For those who have already learnt the concepts, they will be your reminders.

**How to solve a problem in IST?**

There are many patterns to solve problems in information science and technology. The pattern in the picture below is one of the most common patterns.



In my opinion, the most important and the hardest step of the whole flowchart is the first step. You have to transform what you have into some mathematical formulas with the clear goal here. That is not easy to do, and is more art than science. During this course, you will learn this skill by a lot of examples.

Step 2 is when you try to solve your mathematical model using several algorithmic toolboxes you have learnt during undergraduate courses, such as divide and conquer, dynamic programming, or linear programming. Because we believe that most of IST problems "cannot be solved", we would like to skip this step in this course.

When you want to claim that a problem cannot be solved, there is a way to state that formally, called as "NP-Hardness". To make everyone believe that a problem cannot be solved, people usually try to prove that the problem is NP-hard. That is Step 3, which we will discuss on next week.

There are several ways to cope with a problem that cannot be solved. One of the most conventional ways is to use approximation algorithms (Step 4), which is the main focus of this course.

Approximation algorithms can solve almost all problems, but there are cases where we cannot find any approximation algorithm for our problems. It is possible to claim that no one can do that formally. That would be done in Step 5. If no approximation algorithm can solve our problems, we may try to get back to Step 1 and reformulate the problem there.

**What is online algorithm?**

Another main topic of this class is online algorithm, where you have to produce some "output" while you have not got a whole input. The most well-known problem is called as "secretary problem". We want to have just one secretary, and we have a number of applications for the job. We interview all applicants, but, just after each interview, we have to tell the applicants immediately if they are hired. If we hire one secretary, we cannot fire him/her. So, we cannot interview more applicants, although there might be some with a better capability.

Those kind of questions happen a lot in IST, and we will try to address those problems in this course.

**Optimization Models**

Now, it is the time to work on the first step in our flow chart. We have a problem in IST and we want to solve the problem using some theoretical CS concepts. To kick start a whole process, we have to answer the following 4 questions:

1) What do we have in real world? What are important for our problem? [input]
2) What do we want to have? [output]
3) What the output can be, what it cannot be? [constraint]
4) What is the most desirable output? [objective function]

After we have some answers for the 4 questions, we have to transform them into some mathematical formulations. Then, we can devise theoretical tools for those mathematical formulations.

Let us try to illustrate you how it works by a toy example (which is unfortunately not an IST problem). Consider the following problems:

- We can eat only Gyu-don and Melon-pan
- Gyu-don and Melon-pan contain different amounts of carbohydrates and proteins

This is what you have in real world. It is your task to find the "best" way to consume Gyu-don and Melon-pan. We can formulize them by more than one ways, but let try to minimize our cost of living. To do that, we might formulize:

Input: We can look in Google to find how much carbohydrates and proteins that Gyu-don and Melon-pan give. Also, we can look there how much carbohydrates and proteins we should take every day. We can go to some places to check the price of Gyu-don and Melon-pan.

Output: We want to find how much Gyu-don and Melon-pan we should eat in each day.

Constraint: We want to have enough proteins and carbohydrates in every day.

Objective Function: We want to find the amounts that minimize the cost we have to pay in each day.

After we have some rough ideas of our optimization models, we will now formulate it into some mathematical formulations. One way to do is as follows:

Input: Carbohydrate amount that a Gyu-don gives $a_{c,g} \in \mathbb{R}_{\geq 0}$

Carbohydrate amount that a Melon-pan gives $a_{c,m} \in \mathbb{R}_{\geq 0}$

Protein amount that a Gyu-don gives $a_{p,g} \in \mathbb{R}_{\geq 0}$

Protein amount that a Melon-pan gives $a_{p,m} \in \mathbb{R}_{\geq 0}$

Amount of carbohydrate that we should have per day $b_c \in \mathbb{R}_{\geq 0}$

Amount of protein that we should have per day $b_p \in \mathbb{R}_{\geq 0}$

Price of a Gyu-don $c_g \in \mathbb{R}_{\geq 0}$

Price of a Melon-pan $c_m \in \mathbb{R}_{\geq 0}$

Output: Amount of Gyu-don we take in one day $x_g \in \mathbb{R}_{\geq 0}$

Amount of Melon-pan we take in one day $x_m \in \mathbb{R}_{\geq 0}$

Constraint: $a_{c,g}x_g + a_{c,m}x_m \geq b_c$

$a_{p,g}x_g + a_{p,m}x_p \geq b_p$

Objective Function: Minimize $c_g x_g + c_m x_m$

We can leave only mathematical details without losing any information.

Input: $a_{c,g}, a_{c,m}, a_{p,g}, a_{p,m}, b_c, b_p, c_g, c_m \in \mathbb{R}_{\geq 0}$

Output: $x_g, x_m \in \mathbb{R}_{\geq 0}$

Constraint: $a_{c,g}x_g + a_{c,m}x_m \geq b_c$

$a_{p,g}x_g + a_{p,m}x_p \geq b_p$

Objective Function: Minimize $c_g x_g + c_m x_m$

The above is what we will call as "optimization model" in this course. When we can arrive to something in this form, we are ready to take off. We can apply several algorithmic and theoretical techniques to solve the problem. As most of you can guess, we can use a technique called as "linear programming" to solve the problem.

**Linear Programming (LP)**

LP is a very powerful algorithmic toolbox. In fact, it is shown that "LP can solve all problems that can be solved [1]". The formulation is in the following form:

Input: Matrix **A**, vectors **b**, **c**

Output: Vector **x**

Constraint: $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$

Objective Function: Minimize $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}$

It is known that we can use a method called "the interior-point method" to solve LP [2]. However, a heuristic algorithm, called as "simplex method", although cannot solve LP for some inputs, is known to be faster in practice. There are a lot of libraries developed to solve LPs. In my experiments, CPLEX®, developed by IBM, is one of the fastest libraries.

Let us get back to our previous example. We can reformulate the "Gyu-don & Melon-pan problem" to the following:

Input: $\mathbf{A} = \begin{bmatrix} a_{c,g} & a_{c,m} \\ a_{p,g} & a_{p,m} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_c \\ b_p \end{bmatrix}, \mathbf{c} = \begin{bmatrix} c_g \\ c_m \end{bmatrix}$

Output: $\mathbf{x} = \begin{bmatrix} x_g \\ x_m \end{bmatrix}$

Constraint: $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$

Objective Function: Minimize $\mathbf{c}^{T} \cdot \mathbf{x}$

After the reformulation, it is clear that we can use LP algorithm to solve the problem. This course will give you examples how problems are reformulated. We hope that we will get the skill by taking this course.

**What do I mean by "solve"?**

Suppose that the size of our input file is $n$ bits. In this course, we will say that "we can solve an optimization model" if we can find an "algorithm of which the running time is no more than a polynomial function of $n$". We will call such kind of algorithms as "polynomial-time algorithm" during this course.

The polynomial-time algorithms do not always "solve" problems in practice. Consider the case when the running time is somewhere around $n^{1000}$, which is still a polynomial function of $n$. The running time we need for 10-bit input is $10^{1000}$, which is more than the number of atoms in the world. However, in our experience, when one can find an algorithm with the $n^{1000}$ running time, we usually can reduce the running time to $n^5$ or $n^6$.

In the last few years, when we are in big data era, the input size $n$ becomes very large. The number is sometimes even larger than $10^{12}$. When we have such a large input, we cannot even scan everything we have there. The algorithm with running time = $n$ is not acceptable anymore, and we have to find an algorithm with even smaller running time.

Because of that, in big data era, we have to find an alternative definition for "solve". Those theories are still young and we still have a lot to explore. We will sometimes mention them during this course.

I also want to mention that big data is an opportunity for online algorithms. Because we cannot scan a whole input, we have to output the best things we "learn" from the partial information. Online algorithms can do that, so it is fit to the setting.

**Exercises**

At international center for information science and technology, we have a mission to make all communications between students possible. Unfortunately, there is still a language barrier between students. Let a language ability of Student 1 and Student 2 be $\ell_1$ and $\ell_2$ respectively. By a research result, we found that a communication between them will happen when $\ell_1 + \ell_2$ is no smaller than a given integer $\alpha \geq 1$. We know the current language ability of all students. We increase the language ability of students so that all pairs of students can communicate together. However, increasing students' language ability is not free. If we increase a cost of a student from $\ell$ to $\ell'$, it will cost us $10000 \cdot (\ell' - \ell)$ yen. We want to minimize our cost.

Question 1: Suppose that we have 3 students. The language abilities of Student 1, 2, 3 are 2, 3, 1 respectively, and $\alpha$ is 5. Which students should we improve their language ability? And, how much should we improve?

Question 2: What is the optimization model for solving Question 1?

Question 3: Construct a linear program to solve the optimization model in Question 2.

**Hand-on exercises**

CPLEX® is free only for academics and installing it is not a very easy task. The easiest way to use the package software is to use the online tool at https://neos-server.org/neos/solvers/milp:CPLEX/LP.html.

Load a file with the following text to the file box "LP file" at the "Web submission form".

```
Maximize
x1 + 2x2 + 3x3
st
-x1 + x2 + x3 <= 20
x1 - 3x2 + x3 <= 30
x1 <= 40
end
```

Question 4: Which LP we are aiming to solve by the above text file? What is matrix **A**, vectors **b**,**c** in this case. What are inputs, outputs, constraints, and objective functions here?

Also, load a file with the following text to the file box "Display file" at the "Web submission form". The command tells CPLEX to give the value of `x1`, `x2`, and `x3` that satisfy the constraint and optimize the objective function.

```
display solution variables -
```

You have to put your e-mail address here. This is to notify you the results in case that it takes a long time to solve the linear program and your web browser terminates before the solver finishes. I have never received a spam mail from this website.

Question 5: What is the value of `x1`, `x2`, and `x3` that satisfy the constraint and optimize the objective function? What do you have from the solver?

Question 6: Create an LP file to solve the "Gyu-don Melon-pan problem" when $a_{c,g} = 30, a_{c,m} = 50, a_{p,g} = 40, a_{p,m} = 60, b_c = 200, b_p = 200, c_g = 300, c_m = 500$. What is the solution that you have from LP?

Question 7: Discuss why the solution in Question 6 is not practical.

**References**

[1] S. Miyano, S. Shiraishi, and T. Shoudai, "*A List of P-Complete Problems*", Kyushu University, RIFIS-TR-CS-17, 1990.

[2] N. Karmarkar, "*A new polynomial-time algorithm for linear programming*", Combinatorica, Vol. 4, No. 4. Pages 373-395. 1984.