Advanced Topics in Operations Research Notes

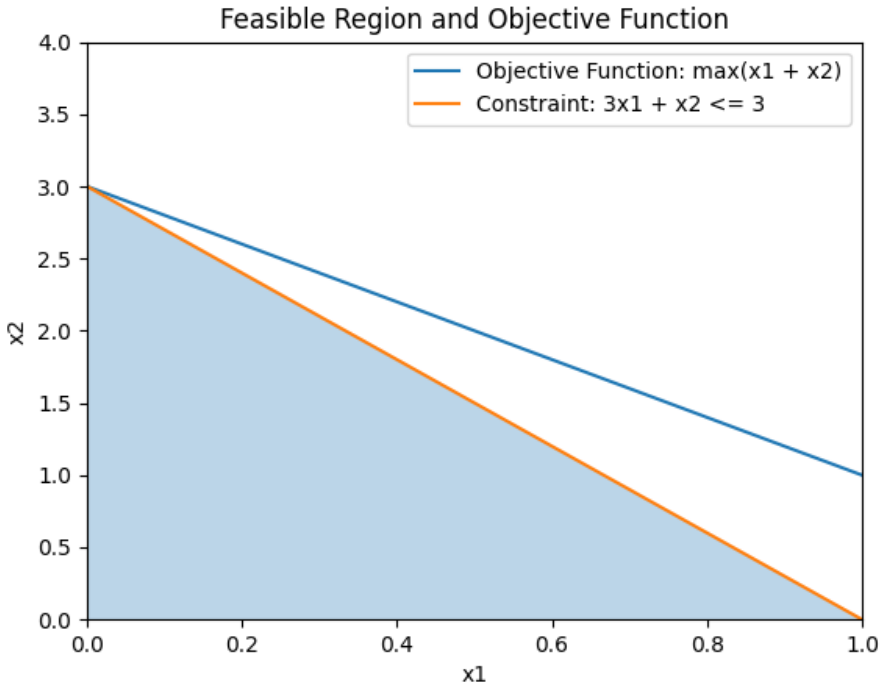
Andres Espinosa

September 4, 2024

Contents

1 Wednesday 08/28/2024

$$max\ x_1 + x_2 \tag{1}$$
$$s.t.\ 3x_1 + x_2 \leq 3 \tag{2}$$
$$x_1 \geq 0, x_2 \geq 0 \tag{3}$$



x^* is an extreme point if we cannot find $x_1, x_2 \in S$, such that $x_1 \neq x_2$ and $x^* = \lambda x_1 + (1-\lambda)x_2$ for some $\lambda \in (0, 1)$
For an n-dimensional problem, there are n-many constraints that are active/binding and their coefficients are linearly independent.
Optimal solutions can be found that are not basic feasible solutions when constraints and objective functions lie on the same plane.

2 Friday 08/30/2024

2.1 Complexity Introduction

Three elements for complexity analysis:

- Model or problem formulation - The known part of a problem. Includes the formulation and problem descriptions.
- Oracle - The smallest computing unit, whose details inside can be ignored.
- Target outcome - When to stop

Two types of complexity:

- Analytical complexity - The number of calls of the oracle which is necessary to solve a given problem formulation up to accuracy ϵ
- Numerical complexity - The number of arithmetic operations which is necessary to solve a given problem formulation up to accracy ϵ

2.2 P, NP, NP-hard

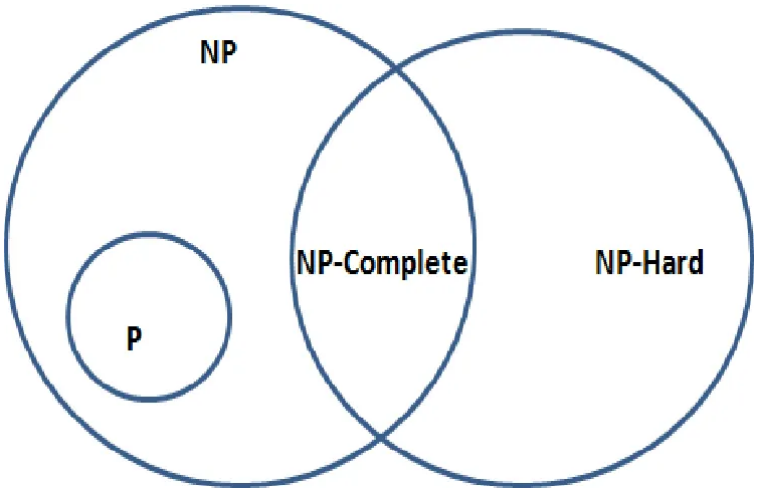
A **problem** is a function $F : I \rightarrow B$, where I is the set of instances encoded as strings of characters and B is the set of problem outputs.

For example, linear programming in $\mathbb{R}^{mxn}$ is a problem where the instance parameters are (m, n, A, c, b) and the output B is the optimal solution found to the objective function.
Another example, making a decision takes in an instance I and outputs a decision of yes or no.

$$I = \textit{Problem-specific information parameters}, B \in (yes, no) \tag{4}$$

3 Wednesday 09/04/2024

3.1 P, NP, NP-complete, and NP-hard



A **problem** is a function $F : I \to B$, where I is the set of instances encoded as strings of characters and B is the set of problem outputs.

A mapping from I to B by taking in the input parameters of what the specific problem actually entails, and then the solution or answer is parametrized by B.

For example, linear programming in $\mathbb{R}^{mxn}$ is a problem where the instance parameters are (m, n, A, c, b) and the output B is the optimal solution found to the objective function.

Another example, making a decision takes in an instance I and outputs a decision of yes or no.

$$I = \textit{Problem-specific information parameters}, B \in (yes, no) \tag{5}$$

3.1.1 Turing machine

A turing machine is a model of how a computer processes a program. There is a *tape* and *finite-state controller*

- **Tape**: A memory unit divided into tape cells. Each tape cell stores one symbol chosen from a finite set of symbols. The tape head is the sole cell currently processed by the controller.
- **Finite-state controller**: A collection of finite states and rules where at each time step, the controller takes an action of erase, move, update cell, and update state.

There is only one unique rule applied for each symbol-state combination

A turing machine is said to be able to solve a problem $F : I \to B$, if given any instance $x \in I$, the turing machine eventually produces the solution B and stops.

My understanding of this is that a turing machine has a controller that goes and selects individual cells based off of the symbol and then makes an action to change the state based off of a set of symbol-state pairs.

3.1.2 P-problem

The class P of problems is defined to be those problems $F : I \rightarrow yes, no$ such that a Turing machine M can compute F, and the number of steps required by M for this computation is bounded by $p(n)$ where p is a polynomial and n is the length of the input.

In other words, a problem P can be solved in polynomial time by a turing machine.

F is a decision problem

NP problem: Problems whose solution, if it is "yes", have proofs verifiable in polynomial time by a deterministic Turing machine.

P problems can be solved in polynomial time, NP problems are problems that can be verified in polynomial time. A non-deterministic Turing machine would be able to solve an NP problem in polynomial time. *A non-deterministic Turing machine is defined as the relaxation of the "unique rule per symbol-state" constraint.*
For example, in the exercise done in class, instead of having the unique rule of "if no one is to the right, say 'yes'", you could have a non-deterministic rule where you have a 50% chance of saying 'no' or 'yes'. Putting randomness in the rule would make it a non-deterministic machine.

3.1.3 NP-complete

A decision problem $F : I \rightarrow yes, no$ can be polynomial transformed into another decision problem $G : I' \rightarrow yes, no$ if there exists a function $f : I \rightarrow I'$. The function f can be computed by a polynomial-time Turing machine, and f has the property that $F(x) = yes \iff G(f(x)) = yes$

Suppose $F : I \rightarrow yes, no$ is a decision problem that lies in NP, and suppose that every problem in NP can be polynomially transformed to F . Then, F is said to be NP-complete