Decision Problems,

The complexity classes P & NP

Decision Problem

- A **decision problem** is a problem whose answer to each instance is either "yes" or "no".
  - Reachability – Given a graph $G = (V, E)$ and $s, t \in V$, determine if there exists an s-t path in G.
  - Connectivity – Given a graph G, determine if G is connected.
  - Partition – Given $A = \{a_1, a_2, \ldots, a_n\}$, determine if there exists a way to partition A into two equal-sum subsets.
  - etc.

The complexity class **P**

- The complexity class **P** consists of the problems that can be solved efficiently in time polynomial in its input length.
  - A problem Π is in P, if there exists an algorithm that computes the answer for each input instance in polynomial-time.
  - For example,
    Connectivity, Reachability, Shortest-Path, Maximum Flow, etc.

The complexity class **NP**

- (Slightly informally speaking)
  The complexity class **NP** consists of the problems whose answers can be verified efficiently in time polynomial in its input length.

  – A problem Π is in NP,
  if there exists an algorithm that correctly verifies any attempt of a “yes”-claim for each instance in polynomial-time.

The complexity class **NP**

- More formally speaking, a problem Π is in NP, if there exists a polynomial-time algorithm A_Π such that,
  - For each "yes"-instance I of Π, there exists a (proof) y such that, the algorithm A_Π accepts the input (I, y).
  - For each "no"-instance I of Π,
    the algorithm A_Π rejects the input (I, y) for all possible y.

The complexity class **NP**

- More formally speaking, a problem Π is in NP, if there exists a polynomial-time algorithm A_Π such that,
  - For each “yes”-instance I of Π, there exists a (proof) y such that, the algorithm A_Π accepts the input (I, y).
  - For each “no”-instance I of Π,
    the algorithm A_Π rejects the input (I, y) for all possible y.
- Informally speaking, a problem is in NP, if there exists a proof system for its instances that can be correctly verified in polynomial-time.

The complexity class **NP**

- For example, the following problems are in NP.
  - Reachability – Given a graph $G = (V, E)$ and $s, t \in V$, determine if there exists an s-t path in G.

    The proof system can be a valid s-t path in G.

    - For a “yes”-instance, there exists an s-t path, and it can be verified in polynomial-time.
    - For a “no”-instance, there exists no s-t path, and there exists no s-t path that can fool the algorithm.

The complexity class **NP**

- For example, the following problems are in NP.
  - Partition – Given $A = \{a_1, a_2, \ldots, a_n\}$, determine if there exists a way to partition A into two equal-sum subsets.

    The proof system can be a valid partition A_1, A_2 of A.

    - The validity of A_1, A_2 can be checked in polynomial-time, i.e., whether or not

$$\sum_{i \in A_1} a_i = \sum_{i \in A_2} a_i \, .$$

The complexity class **NP**

- For example, the following problems are in NP.
  - Connectivity – Given a graph G, determine if G is connected.

    The proof system can be a DFS-tree (resulted by a traversal) of G.

    - We can verify the validity of the DFS tree in polynomial-time and check if it contains all the vertices.

An Alternative Way to View **P**

- Provided the definition & interpretation of NP,
  the following is a natural way to view the complexity class P.

  – A problem Π is in P,
    if there exists an algorithm that writes a valid proof
    for each input instance (including “yes”- and “no”-instances)
    in polynomial-time.

> Imagine that…,
> for an algorithm that solves the problem in polynomial-time,
> the process of its computation process is exactly a proof that can be verified.

P versus **NP**

- From the definitions,
  it is easy to see that $\boldsymbol{P} \subseteq \boldsymbol{NP}$.

- Whether or not $\boldsymbol{NP} \overset{?}{\subseteq} \boldsymbol{P}$ is <u>*a major open problem*</u> in computer science.
  - From our previous interpretations,
    the open question states:
    <u>*Are problems that are easy to verify also easy to prove*</u>?
  - Alternatively,
    <u>***Is writing proofs as easy as verifying proofs***</u>**?**

Basic Concepts & Definitions

Optimization Problems

For example,
the shortest $s - t$ path problem.

- An ***optimization problem***, Π, consists of the following:
  - A set of valid instances, D_Π.
    The size (length) of an instance $I \in D_\Pi$, denoted $|I|$, is the number of bits needed to write I in binary representation.
  - Each instance $I \in D_\Pi$ has a nonempty set of feasible solutions $S_\Pi(I)$.
  - An objective function that assigns a value to each pair (I, s), where $s \in S_\Pi(I)$.
  - Π is specified to be either a minimization or a maximization problem.
- The goal of an ***optimization problem*** Π is to pick the ***"best"*** solution from the feasible solutions of the input instance $I \in D_\Pi$.

A graph $G = (V, E)$ with $s, t \in V$.

The set of all $s - t$ paths.

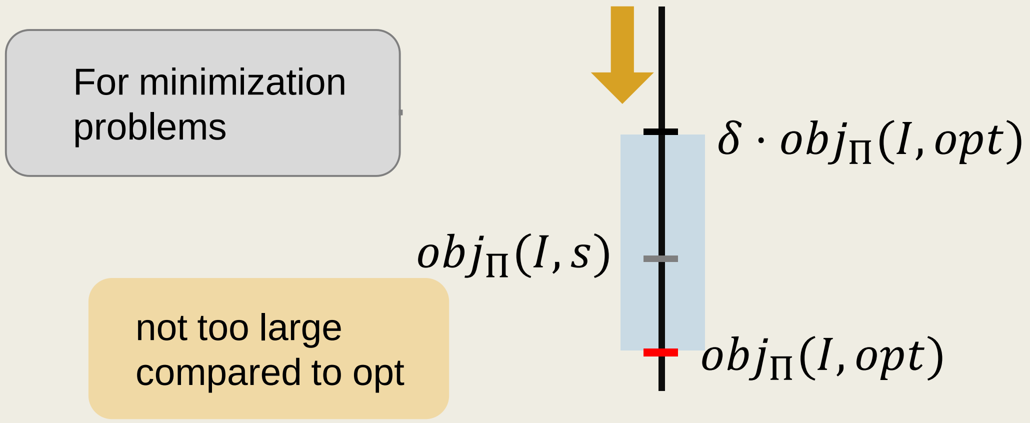
The length of the paths.

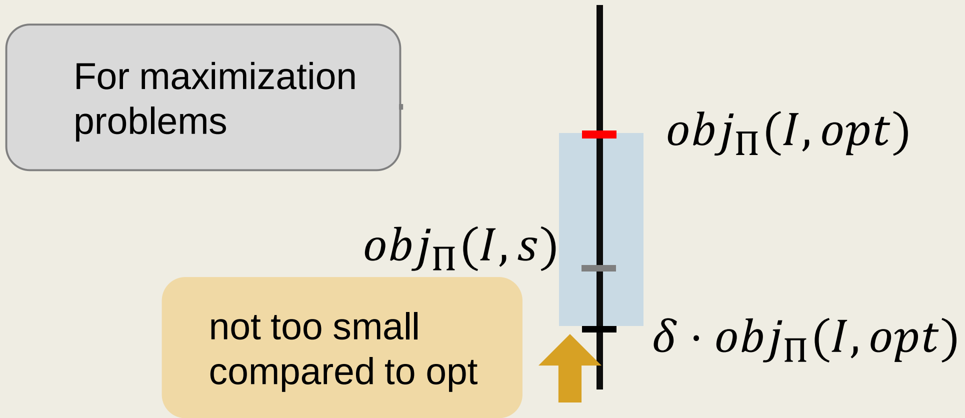
Compute the shortest $s - t$ path in G.

Approximate Solutions for Optimization Problems

- A feasible solution s for an instance I of an optimization problem Π is said to be a δ***-approximate solution*** if
  - $obj_{\Pi}(I, s) \leq \delta \cdot obj_{\Pi}(I, opt)$, if Π is a minimization problem,
  - $obj_{\Pi}(I, s) \geq \delta \cdot obj_{\Pi}(I, opt)$, if Π is a maximization problem,

where obj_{Π} is the objective function of Π and opt is an optimal solution of I.





Approximation Algorithms for Optimization Problems

- An algorithm A is said to be a δ***-approximation algorithm*** for an optimization Π if, for each instance I of Π, the algorithm A produces a δ-approximate solution for I and the *running time of* A *is bounded by a polynomial of* $|I|$.
  - δ is referred to as the ***approximation ratio***, ***approximation factor***, or, ***approximation guarantee***, of A.

For any *given, fixed, constant* ϵ,
we have just seen a $(1-\epsilon)$-approximation algorithm for the Knapsack problem .

Approximation Schemes

- An algorithm A is said to be an approximation scheme for an optimization Π if, on input instance I and error parameter $\epsilon > 0$, the algorithm A produces
  - a $(1 + \epsilon)$-approximate solution for I, if Π is a minimization problem,
  - a $(1 - \epsilon)$-approximate solution for I, if Π is a maximization problem.

- An approximation scheme A is said to be
  - A ***polynomial-time approximation scheme***, abbreviated **PTAS**, if its running time is bounded by a polynomial in $|I|$.
  - A ***fully polynomial-time approximation scheme***, abbreviated **FPTAS**, if its running time is bounded by a polynomial in $|I|$ and $1/\epsilon$.

Some Fundamental Concepts

Decision Problem vs Optimization Problem

- A ***decision problem*** is a problem whose answer to each instance is either “yes” or “no.”
  - Ex. Reachability, Connectivity, etc.

- An ***optimization problem*** is a problem whose goal is to pick the ***“best”*** solution from a set of feasible solutions.
  - We have an *objective function* which associates each feasible solution a value.
  - Ex. Shortest-Path, MST, Knapsack, etc.

Decision Version of Optimization Problems

- An optimization problem can also be formulated in a ***decision form***.
  - Instead of optimizing the solution, we ask,
    "is there a solution whose value is at least (at most) c,
    for a given constant c ? "

- Then…
  - Optimization form implies Decision form.
  - Interestingly, ***Decision form also implies Optimization form!***
    - When the objective is polynomially bounded.

By applying binary search.

Decision Version of Optimization Problems

- Then…
  - Optimization form implies Decision form.
  - Interestingly, ***Decision form also implies Optimization form***!
    - When the objective is polynomially bounded.
    - i.e., we can use decision version to solve the optimization version.

- So, in general, for optimization problems,
  we don’t need to bother with their decision forms.

NP-hard and NP-complete Problems

- A problem is said to be ***NP-hard*** if it can be used to **"solve"** ***all problems*** **in NP**.
  - More formally speaking, each problem in NP can be transformed (reduced) to this problem.
  - In some sense,
    an NP-hard problem is ***<u>at least as hard as</u>*** all the problems in **NP**.
- An NP-hard problem is said to be ***NP-complete*** if it is also in **NP**.