3-9 问题的形式化描述

程龚

你理解布尔函数的这些不同“表示”了吗?

Definition 2.2.3.3. *A* **Boolean variable** *is any symbol to which one can associate either of the values* 0 *or* 1.

Let $X = \{x_1, \ldots, x_n\}$ *be a set of Boolean variables for some* $n \in \mathbb{N}$. *A* **Boolean function over** X *is any mapping* f *from* $\{0,1\}^n$ *to* $\{0,1\}$. *One denotes* f *by* $f(x_1, x_2, \ldots, x_n)$ *if one wants to call attention to the names of its variables.*

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Truth table

$N^1(f(x_1, x_2, x_3)) = \{(0,0,0), (0,1,1), (1,0,0), (1,1,0)\}$, and
$N^0(f(x_1, x_2, x_3)) = \{(0,0,1), (0,1,0), (1,0,1), (1,1,1)\}$.

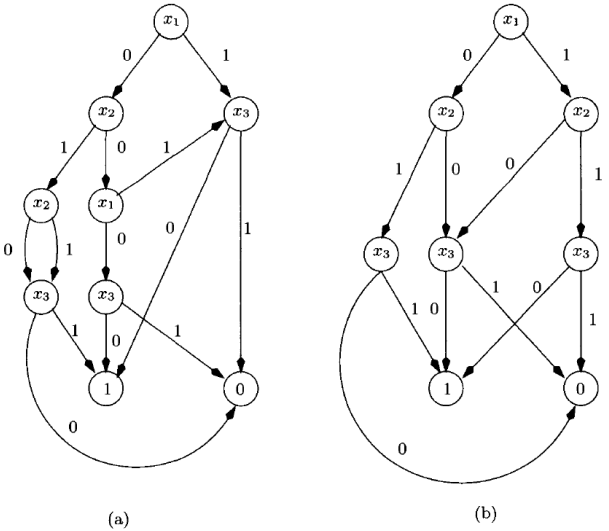
Input assignment

$$(\overline{x}_1 \wedge \overline{x}_2 \wedge \overline{x}_3) \vee (\overline{x}_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x}_2 \wedge \overline{x}_3) \vee (x_1 \wedge x_2 \wedge \overline{x}_3)$$

Complete DNF

$$(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$$

Complete CNF



Branching program

对于同一个对象，我们为什么需要不同的“表示”？

算法问题的表示

The main goal of this section is to give definitions of notions that are sufficient for fixing the representation of some input data and thus to precisely formalize the definitions of some fundamental algorithmic problems. We also need the terms defined here for the abstract considerations of the complexity theory in Section 2.3.3 and for proving lower bounds on polynomial-time inapproximability in Section 4.4.2.

你理解这些概念了吗？

- Alphabet（字母表）
- Symbol（符号）
- Word（词）
  - concatenation、 prefix/suffix/subword、 $\boldsymbol{Number(u)} = \sum_{i=1}^{n} u_i \cdot 2^{i-1}$
- Language（语言）
  - concatenation

你理解这些概念了吗？

- Alphabet（字母表）
- Symbol（符号）
- Word（词）
  - concatenation、 prefix/suffix/subword、 $\boldsymbol{Number(u)} = \sum_{i=1}^{n} u_i \cdot 2^{i-1}$
- Language（语言）
  - concatenation

- 你能设计一种语言来编码以下信息吗？
  你设计的字母表、词、语言分别是什么？
  - 全班同学的一次问求期末考试成绩
  - 一张图片
  - 一段视频
  - 你能自己再举一个日常生活中的例子吗？

你理解图的这种表示了吗？它合理吗？

One can use the alphabet $\{0, 1, \#\}$ to code graphs. If $M_G = [a_{ij}]_{i,j=1,\ldots,n}$ is an adjacency matrix of a graph G of n vertices, then the word

$$a_{11}a_{12}\ldots a_{1n}\#a_{21}a_{22}\ldots a_{2n}\#\ldots\#a_{n1}a_{n2}\ldots a_{nn}$$

can be used to code G.

你理解这些概念了吗？

Definition 2.3.1.10. *Let* $\Sigma = \{s_1, s_2, \ldots, s_m\}$, $m \geq 1$, *be an alphabet, and let* $s_1 < s_2 < \cdots < s_m$ *be a linear ordering on* Σ. *We define the* **canonical ordering** *on* Σ^* *as follows. For all* $u, v \in \Sigma^*$,

$$\begin{aligned} u < v \text{ if } & |u| < |v| \\ \text{or } & |u| = |v|, u = xs_iu', \text{ and } v = xs_jv' \\ & \text{for some } x, u', v' \in \Sigma^*, \text{ and } i < j. \end{aligned}$$

- 我们为什么需要一种排序规则？
- 你能定义一种不同的排序规则吗？

你理解这段话了吗？

Thousands of algorithmic problems classified according to different points of view are considered in the literature on algorithmics. In this book we deal with hard problems only. We consider a problem to be hard if there is no known deterministic algorithm (computer program) that solves it efficiently. Efficiently means in a low-degree polynomial time. Our interpretation of hardness here is connected to the current state of our knowledge in algorithmics rather than to the unknown, real difficulty of the problems considered. Thus, a problem is hard if one would need years or thousands of years to solve it by deterministic programs for an input of a realistic size appearing in the current practice. This book provides a handbook of algorithmic methods that attack hard problems. Thousands of problems of great practical relevance are very hard from this point of view. Fortunately, we do not need to define and to consider all of them. There are some crucial, paradigmatic problems such as the traveling salesperson problem, linear (integer) programming, set cover problem, knapsack problem, satisfiability problem, and primality testing that are pattern problems in the sense that solving most of the hard problems can be reduced to solving some of the paradigmatic problems.

你理解这段话了吗？

Every algorithm (computer program) can be viewed as an execution of a mapping from a subset of Σ_1^* to Σ_2^* for some alphabets Σ_1 and Σ_2. So, every (algorithmic) problem can be considered as a function from Σ_1^* to Σ_2^* or as a relation on $\Sigma_1^* \times \Sigma_2^*$ for some alphabets Σ_1 and Σ_2. We usually do not need to work with this kind of formalism because we consider two classes of problems only – decision problems (to decide for a given input whether it has a prescribed property) and optimization problems (to find the "best" solution from the set of solutions determined by some constraints). In what follows we

- 其中的word分别是什么？

你理解这些概念了吗？

Definition 2.3.2.1. *A* **decision problem** *is a triple* (L, U, Σ) *where* Σ *is an alphabet and* $L \subseteq U \subseteq \Sigma^*$. *An algorithm* A **solves** (**decides**) *the decision problem* (L, U, Σ) *if, for every* $x \in U$,

(i) $A(x) = 1$ *if* $x \in L$, *and*
(ii) $A(x) = 0$ *if* $x \in U - L$ $(x \notin L)$.

- 其中的word是什么？

你理解这些概念了吗?

Definition 2.3.2.1. *A* **decision problem** *is a triple* (L, U, Σ) *where* Σ *is an alphabet and* $L \subseteq U \subseteq \Sigma^*$. *An algorithm* A **solves** (**decides**) *the decision problem* (L, U, Σ) *if, for every* $x \in U$,

(i) $A(x) = 1$ *if* $x \in L$, *and*
(ii) $A(x) = 0$ *if* $x \in U - L$ $(x \notin L)$.

- 其中的word是什么?
- 你理解这段话了吗?

An equivalent form of a description of a decision problem is the following form that specifies the input-output behavior.

Problem (L, U, Σ)

Input: An $x \in U$.
Output: "yes" if $x \in L$,
"no" otherwise.

For many decision problems (L, U, Σ) we assume $U = \Sigma^*$. In that case we shall use the short notation (L, Σ) instead of (L, Σ^*, Σ).

你理解这个判定问题了吗？这里的L是什么？

PRIMALITY TESTING.

Informally, primality testing is to decide, for a given positive integer, whether it is prime or not. Thus, primality testing is a decision problem $(\text{PRIM}, \Sigma_{bool})$, where

$$\text{PRIM} = \{w \in \{0,1\}^* \mid Number(w) \text{ is a prime}\}.$$

Another description of this problem is

Primality testing

Input: An $x \in \Sigma^*_{bool}$
Output: "yes" if $Number(x)$ is a prime,
"no" otherwise.

你理解这段话了吗？

One can easily observe that primality testing can also be considered for other integer representations. Using $\Sigma_k = \{0, 1, 2, \ldots, k-1\}$ and the k-ary representation of integers we obtain $(\text{PRIM}_k, \Sigma_k)$, where

$$\text{PRIM}_k = \{x \in \Sigma_k^* \mid x \text{ is the } k\text{-ary representation of a prime}\}.$$

From the point of view of computational hardness it is not essential whether we consider $(\text{PRIM}, \Sigma_{bool})$ or $(\text{PRIM}_k, \Sigma_k)$ for some constant k because one has efficient algorithms for transferring any k-ary representation of an integer to its binary representation, and vice versa. But this does not mean that the representation of integers does not matter for primality testing. If one represents an integer n as

$$\#bin(p_1)\#bin(p_2)\#\ldots\#bin(p_l)$$

over $\{0, 1, \#\}$, where $n = p_1 \cdot p_2 \cdot \cdots \cdot p_l$ and p_is are the nontrivial prime factors of n, then the problem of primality testing becomes easy. This sensibility of hardness of algorithmic problems according to the representation of their inputs is sometimes the reason for taking an exact formal description of the problem that fixes the data representation, too. For primality testing we always consider $(\text{PRIM}, \Sigma_{bool})$ as the formal definition of this decision problem.

你理解这个判定问题了吗？这里的L是什么？

EQUIVALENCE PROBLEM FOR POLYNOMIALS.

The problem is to decide, for a given prime p and two polynomials $p_1(x_1, \ldots, x_m)$ and $p_2(x_1, \ldots, x_m)$ over the field $\mathbb{Z}_p$, whether p_1 and p_2 are equivalent, i.e., whether $p_1(x_1, \ldots, x_m) - p_2(x_1, \ldots, x_m)$ is identical 0. The crucial point is that the polynomials are not necessarily given in a normal form such as

$$a_0 + a_1x_1 + a_2x_2 + a_{12}x_1x_2 + a_1^2x_1^2 + a_2^2x_2^2 + \cdots$$

but in an arbitrary form such as

$$(x_1 + 3x_2)^2 \cdot (2x_1 + 4x_4) \cdot x_3^2.$$

A normal form may be exponentially long in the length of another representation and so the obvious way to compare two polynomials by transferring them to their normal forms and comparing their coefficients is not efficient.

We omit the formal definition of the representation of polynomials in an arbitrary form over the alphabet $\Sigma_{pol} = \{0, 1, (,), \exp, +, \cdot\}$, because it can be done in a similar way as how one represents formulae over Σ_{logic}. The equivalence problem for polynomials can be defined as follows.

EQ-POL

Input: A prime p, two polynomials p_1 and p_2 over variables from $X = \{x_1, x_2, \ldots\}$.
Output: "yes" if $p_1 \equiv p_2$ in the field $\mathbb{Z}_p$,
"no" otherwise.

你理解这个判定问题了吗？这里的L是什么？

EQUIVALENCE PROBLEM FOR ONE-TIME-ONLY BRANCHING PROGRAMS.

The equivalence problem for one-time-only branching programs, EQ-1BP, is to decide, for two given one-time-only branching programs B_1 and B_2, whether B_1 and B_2 represent the same Boolean function. One can represent a branching program in a similar way as a directed weighted graph[28] and so we omit the formal description of branching program representation.[29]

EQ-1BP

Input: One-time-only branching program B_1 and B_2 over a set of Boolean variables $X = \{x_1, x_2, x_3, \ldots\}$.

Output: "yes" if B_1 and B_2 are equivalent (represent the same Boolean function),
"no" otherwise.

你理解这个判定问题了吗？这里的L是什么？

SATISFIABILITY PROBLEM.

The satisfiability problem is to decide, for a given formula in the CNF, whether it is satisfiable or not. Thus, the **satisfiability problem** is the decision problem $(\text{SAT}, \Sigma_{logic})$, where

$$\textbf{SAT} = \{w \in \Sigma^+_{logic} \mid w \text{ is a code of a satisfiable formula in CNF}\}.$$

We also consider specific subproblems of SAT where the length of clauses of the formulae in CNF is bounded. For every positive integer $k \geq 2$, we define the ***k*-satisfiability problem** as the decision problem $(k\text{SAT}, \Sigma_{logic})$, where

$$\boldsymbol{k}\textbf{SAT} = \{w \in \Sigma^+_{logic} \mid w \text{ is a code of a satisfiable formula in } k\text{CNF}\}.$$

In what follows we define some decidability problems from graph theory.

你理解这个判定问题了吗？这里的L是什么？

CLIQUE PROBLEM.

The clique problem is to decide, for a given graph G and a positive integer k, whether G contains a clique of size k (i.e., whether the complete graph K_k of k vertices is a subgraph of G). Formally, the **clique problem** is the decision problem $(\text{CLIQUE}, \{0, 1, \#\})$, where

$$\text{CLIQUE} = \{x\#w \in \{0,1,\#\}^* \mid x \in \{0,1\}^* \text{ and } w \text{ represents a graph that contains a clique of size } \mathit{Number}(x)\}.$$

An equivalent description of the clique problem is the following one.

Clique Problem

Input: A positive integer k and a graph G.
Output: "yes" if G contains a clique of size k,
"no" otherwise.

你理解这个判定问题了吗？这里的L是什么？

VERTEX COVER PROBLEM.

The vertex cover problem is to decide, for a given graph G and a positive integer k, whether G contains a vertex cover of cardinality k. Remember that a vertex cover of $G = (V, E)$ is any set S of vertices of G such that each edge from E is incident to at least one vertex in S.

Formally, the **vertex cover problem** (**VCP**) is the decision problem $(\text{VCP}, \{0, 1, \#\})$, where

$$\mathbf{VCP} = \{u\#w \in \{0,1,\#\}^+ \mid u \in \{0,1\}^+ \text{ and } w \text{ represents a graph that contains a vertex cover of size } \mathit{Number}(u)\}.$$

你理解这个判定问题了吗？这里的L是什么？

HAMILTONIAN CYCLE PROBLEM.

The Hamiltonian cycle problem is to determine, for a given graph G, whether G contains a Hamiltonian cycle or not. Remember that a Hamiltonian cycle of G of n vertices is a cycle of length n in G that contains every vertex of G.

Formally, the **Hamiltonian cycle problem** (**HC**) is the decision problem $(\mathrm{HC}, \{0, 1, \#\})$, where

$$\mathbf{HC} = \{w \in \{0,1,\#\}^* \mid w \text{ represents a graph that contains a Hamiltonian cycle}\}.$$

你理解这个判定问题了吗？这里的L是什么？

EXISTENCE PROBLEMS IN LINEAR PROGRAMMING.

Here, we consider problems of deciding whether a given system of linear equations has a solution. Following the notation of Section 2.2.1 a system of linear equations is given by the equality

$$A \cdot X = b,$$

where $A = [a_{ij}]_{i=1,\ldots,m,j=1,\ldots,n}$ is an $m \times n$ matrix, $X = (x_1, x_2, \ldots, x_n)^{\mathsf{T}}$, and $b = (b_1, \ldots, b_m)^{\mathsf{T}}$ is an m-dimensional column vector. The n elements $x_1, x_2, \ldots, x_n$ of X are called unknowns (variables). In what follows we consider that all elements of A and b are integers. Remember, that

$$Sol(A, b) = \{X \subseteq \mathbb{R}^n \mid A \cdot X = b\}$$

denotes the set of all real-valued solutions of the linear equations system $A \cdot X = b$. In what follows we are interested in deciding whether $Sol(A, b)$ is empty or not (i.e., whether there exist a solution to $A \cdot X = b$) for given A and b. More precisely, we consider several specific decision problems by restricting the set $Sol(A, b)$ to subsets of solutions over $\mathbb{Z}^n$ or $\{0, 1\}^n$ only, or even considering the linear equations over some finite fields instead of $\mathbb{R}$. Let

$$\boldsymbol{Sol_S(A, b)} = \{X \subseteq S^n \mid A \cdot X = b\}$$

for any subset S of $\mathbb{R}$.

First of all observe that the problem of deciding whether $Sol(A, b) = \emptyset$ is one of the fundamental tasks of linear algebra and that it can be solved efficiently. The situation essentially changes if one searches for integer solutions or Boolean solutions. Let $\langle A, b\rangle$ denote a representation of a matrix A and a vector b over the alphabet $\{0, 1, \#\}$, assuming all elements of A and b are integers.

The **problem of the existence of a solution of linear integer programming** is to decide whether $Sol_{\mathbb{Z}}(A, b) = \emptyset$ for given A and b. Formally, this decision problem is (SOL-IP, $\{0, 1, \#\}$), where

$$\textbf{SOL-IP} = \{\langle A, b\rangle \in \{0, 1, \#\}^* \mid Sol_{\mathbb{Z}}(A, b) \neq \emptyset\}.$$

你理解这些概念了吗？

Definition 2.3.2.2. *An* **optimization problem** *is a 7-tuple* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, *where*

(i) Σ_I *is an alphabet, called the* **input alphabet** *of* U,
(ii) Σ_O *is an alphabet, called the* **output alphabet** *of* U,
(iii) $L \subseteq \Sigma_I^*$ *is the* **language of feasible problem instances**,
(iv) $L_I \subseteq L$ *is the* **language of the (actual) problem instances of** U,
(v) $\mathcal{M}$ *is a function from* L *to* $Pot(\Sigma_O^*)$,[30] *and, for every* $x \in L$, $\mathcal{M}(x)$ *is called the* **set of feasible solutions** *for* x,
(vi) cost is the **cost function** *that, for every pair* (u, x), *where* $u \in \mathcal{M}(x)$ *for some* $x \in L$, *assigns a positive real number* $cost(u, x)$,
(vii) $goal \in \{minimum, maximum\}$.

For every $x \in L_I$, *a feasible solution* $y \in \mathcal{M}(x)$ *is called* **optimal for** x **and** U *if*

$$cost(y, x) = goal\{cost(z, x) \mid z \in \mathcal{M}(x)\}.$$

For an optimal solution $y \in \mathcal{M}(x)$, *we denote* $cost(y, x)$ *by* $\mathbf{Opt_U(x)}$. U *is called a* **maximization problem** *if* $goal = \text{maximum}$, *and* U *is a* **minimization problem** *if* $goal = minimum$. *In what follows* $\mathbf{Output_U(x)} \subseteq \mathcal{M}(x)$ *denotes the set of all optimal solutions for the instance* x *of* U.

你理解这段话了吗？

The language L is the set of codes of all problem instances (inputs) for which U is well defined. L_I is the set of actual problem instances (inputs) and one measures the computational hardness of U according to inputs of L_I. In general, one can simplify the definition of U by removing L and the definition will work as well as Definition 2.3.2.2. The reason to put this additional information into the definition of optimization problems is that the hardness of many optimization problems is very sensible according to the specification of the set of considered problem instances (L_I). Definition 2.3.2.2 enables one to conveniently measure the increase or decrease of the hardness of optimization problems according to the changes of L_I by a fixed L.

Definition 2.3.2.3. *Let* $U_1 = (\Sigma_I, \Sigma_O, L, L_{I,1}, \mathcal{M}, cost, goal)$ *and* $U_2 = (\Sigma_I, \Sigma_O, L, L_{I,2}, \mathcal{M}, cost, goal)$ *be two optimization problems. We say that* U_1 *is a* **subproblem** *of* U_2 *if* $L_{I,1} \subseteq L_{I,2}$.

- 你能举个图论中的例子吗？

你理解这些概念了吗？

An algorithm A is **consistent** *for U if, for every $x \in L_I$, the output $A(x) \in \mathcal{M}(x)$. We say that an algorithm B* **solves** *the optimization problem U if*

(i) B is consistent for U, and
(ii) for every $x \in L_I$, $B(x)$ is an optimal solution for x and U.

你理解这个优化问题了吗？

Traveling Salesperson Problem (TSP)

Input: A weighted complete graph (G, c), where $G = (V, E)$ and $c : E \to \mathbb{N}$. Let $V = \{v_1, \ldots, v_n\}$ for some $n \in \mathbb{N} - \{0\}$.

Constraints: For every input instance (G, c), $\mathcal{M}(G, c) = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1} \mid (i_1, i_2, \ldots, i_n)$ is a permutation of $(1, 2, \ldots, n)\}$, i.e., the set of all Hamiltonian cycles of G.

Costs: For every Hamiltonian cycle $H = v_{i_1} v_{i_2} \ldots v_{i_n} v_{i_1} \in \mathcal{M}(G, c)$, $cost((v_{i_1}, v_{i_2}, \ldots v_{i_n}, v_{i_1}), (G, c)) = \sum_{j=1}^{n} c(\{v_{i_j}, v_{i_{(j \bmod n)+1}}\})$, i.e., the cost of every Hamiltonian cycle H is the sum of the weights of all edges of H.

Goal: *minimum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Now we define two subproblems of TSP.

The **metric traveling salesperson problem**, **△-TSP**, is a subproblem of TSP such that every problem instance (G, c) of △-TSP satisfies the triangle inequality

$$c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$$

for all vertices u, w, v of G.

The **geometrical traveling salesperson problem** (**Euclidean TSP**) is a subproblem of TSP such that, for every problem instance (G, c) of TSP, the vertices of G can be embedded in the two-dimensional Euclidean space in such a way that $c(\{u, v\})$ is the Euclidean distance between the points assigned to the vertices u and v for all u, v of G. A simplified specification of the set of input instances of the geometric TSP is to say that the input is a set of points in the plane and the cost of the connection between any two points is defined by their Euclidean distance.

Since the two-dimensional Euclidean space is a metric space, the Euclidean distance satisfies the triangle inequality and so the geometrical TSP is a subproblem of △-TSP.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Makespan Scheduling Problem (MS)

Input: Positive integers $p_1, p_2, \ldots, p_n$ and an integer $m \geq 2$ for some $n \in \mathbb{N} - \{0\}$.
{p_i is the processing time of the ith job on any of the m available machines}.

Constraints: For every input instance $(p_1, \ldots, p_n, m)$ of MS,
$\mathcal{M}(p_1, \ldots, p_n, m) = \{S_1, S_2, \ldots, S_m \mid S_i \subseteq \{1, 2, \ldots, n\}$ for $i = 1, \ldots, m$, $\bigcup_{k=1}^{m} S_k = \{1, 2, \ldots, n\}$, and $S_i \cap S_j = \emptyset$ for $i \neq j\}$.
{$\mathcal{M}(p_1, \ldots, p_n, m)$ contains all partitions of $\{1, 2, \ldots, n\}$ into m subsets. The meaning of $(S_1, S_2, \ldots, S_m)$ is that, for $i = 1, \ldots, m$, the jobs with indices from S_i have to be processed on the ith machine}.

Costs: For each $(S_1, S_2, \ldots, S_m) \in \mathcal{M}(p_1, \ldots, p_n, m)$,
$cost((S_1, \ldots, S_m), (p_1, \ldots, p_n, m)) = \max\{\sum_{l \in S_i} p_l \mid i = 1, \ldots, m\}$.

Goal: *minimum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Minimum Vertex Cover Problem (MIN-VCP)

Input: A graph $G = (V, E)$.
Constraints: $\mathcal{M}(G) = \{S \subseteq V \mid$ every edge of E is incident to at least one vertex of $S\}$.
Cost: For every $S \in \mathcal{M}(G)$, $cost(S, G) = |S|$.
Goal: *minimum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Set Cover Problem (SCP)

Input: $(X, \mathcal{F})$, where X is a finite set and $\mathcal{F} \subseteq Pot(X)$ such that $X = \bigcup_{S \in \mathcal{F}} S$.
Constraints: For every input $(X, \mathcal{F})$,
$\mathcal{M}(X, \mathcal{F}) = \{C \subseteq \mathcal{F} \mid X = \bigcup_{S \in C} S\}$.
Costs: For every $C \in \mathcal{M}(X, \mathcal{F})$, $cost(C, (X, \mathcal{F})) = |C|$.
Goal: *minimum*.

Later we shall observe that MIN-VCP can be viewed as a special subproblem of SCP because, for a given graph $G = (V, E)$, one can assign the set S_v of all edges adjacent to v to every vertex v of G.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Weighted Minimum Vertex Cover Problem (WEIGHT-VCP)

Input: A weighted graph $G = (V, E, c)$, $c : V \to \mathbb{N} - \{0\}$.
Constraints: For every input instance $G = (V, E, c)$,
$\mathcal{M}(G) = \{S \subseteq V \mid S \text{ is a vertex cover of } G\}$.
Cost: For every $S \in \mathcal{M}(G)$, $G = (V, E, c)$,
$cost(S, (V, E, c)) = \sum_{v \in S} c(v)$.
Goal: *minimum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Maximum Clique Problem (MAX-CL)

Input: A graph $G = (V, E)$
Constraints: $\mathcal{M}(G) = \{S \subseteq V \mid \{\{u, v\} \mid u, v \in S, u \neq v\} \subseteq E\}$.
$\{\mathcal{M}(G)$ contains all complete subgraphs (cliques) of $G\}$
Costs: For every $S \in \mathcal{M}(G)$, $cost(S, G) = |S|$.
Goal: *maximum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Maximum Cut Problem (MAX-CUT)

Input: A graph $G = (V, E)$.
Constraints:

$$\mathcal{M}(G) = \{(V_1, V_2) \mid V_1 \cup V_2 = V, V_1 \neq \emptyset \neq V_2, \text{and} V_1 \cap V_2 = \emptyset\}.$$

Costs: For every cut $(V_1, V_2) \in \mathcal{M}(G)$,

$$cost((V_1, V_2), G) = |E \cap \{\{u, v\} \mid u \in V_1, v \in V_2\}|.$$

Goal: *maximum*.

The **minimum cut problem** (MIN-CUT) can be defined in the same way as MAX-CUT. The only difference is that the goal of MIN-CUT is minimum.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Simple Knapsack Problem (SKP)

Input: A positive integer b, and positive integers $w_1, w_2, \ldots, w_n$ for some $n \in \mathbb{N} - \{0\}$.

Constraints: $\mathcal{M}(b, w_1, w_2, \ldots, w_n) = \{T \subseteq \{1, \ldots, n\} \mid \sum_{i \in T} w_i \leq b\}$, i.e., a feasible solution for the problem instance $b, w_1, w_2, \ldots, w_n$ is every set of objects whose common weight does not exceed b.

Costs: For each $T \in \mathcal{M}(b, w_1, w_2, \ldots, w_n)$,

$$cost(T, b, w_1, w_2, \ldots, w_n) = \sum_{i \in T} w_i.$$

Goal: *maximum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Knapsack Problem (KP)

Input: A positive integer b, and $2n$ positive integers $w_1, w_2, \ldots, w_n$, c_1, $c_2, \ldots, c_n$ for some $n \in \mathbb{N} - \{0\}$.

Constraints:

$$\mathcal{M}(b, w_1, \ldots, w_n, c_1, \ldots, c_n) = \{T \subseteq \{1, \ldots, n\} \mid \sum_{i \in T} w_i \leq b\}.$$

Costs: For each $T \in \mathcal{M}(b, w_1, \ldots, w_n, c_1, \ldots, c_n)$,

$$cost(T, b, w_1, \ldots, w_n, c_1, \ldots, c_n) = \sum_{i \in T} c_i.$$

Goal: *maximum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Bin-Packing Problem (BIN-P)

Input: n rational numbers $w_1, w_2, \ldots, w_n \in [0,1]$ for some positive integer n.

Constraints: $\mathcal{M}(w_1, w_2, \ldots, w_n) = \{S \subseteq \{0,1\}^n \mid$ for every $s \in S$, $s^{\mathsf{T}} \cdot (w_1, w_2, \ldots, w_n) \leq 1$, and $\sum_{s \in S} s = (1, 1, \ldots, 1)\}$.
{If $S = \{s_1, s_2, \ldots, s_m\}$, then $s_i = (s_{i1}, s_{i2}, \ldots, s_{in})$ determines the set of objects packed in the ith bin. The jth object is packed into the ith bin if and only if $s_{ij} = 1$. The constraint

$$s_i^{\mathsf{T}} \cdot (w_1, \ldots, w_n) \leq 1$$

assures that the ith bin is not overfilled. The constraint

$$\sum_{s \in S} s = (1, 1, \ldots, 1)$$

assures that every object is packed in exactly one bin.}

Cost: For every $S \in \mathcal{M}(w_1, w_2, \ldots, w_n)$,

$$cost(S, (w_1, \ldots, w_n)) = |S|.$$

Goal: *minimum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Maximum Satisfiability Problem (MAX-SAT)

Input: A formula $\Phi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$ over $X = \{x_1, x_2, \ldots\}$ in CNF (an equivalent description of this instance of MAX-SAT is to consider the set of clauses $F_1, F_2, \ldots, F_m$).

Constraints: For every formula Φ over the set $\{x_1, \ldots, x_n\} \subseteq X$, $n \in \mathbb{N} - \{0\}$, $\mathcal{M}(\Phi) = \{0,1\}^n$.
{Every assignment of values to $\{x_1, \ldots, x_n\}$ is a feasible solution, i.e., $\mathcal{M}(\Phi)$ can also be written as $\{\alpha \mid \alpha : X \to \{0,1\}\}$.

Costs: For every Φ in CNF, and every $\alpha \in \mathcal{M}(\Phi)$, $cost(\alpha, \Phi)$ is the number of clauses satisfied by α.

Goal: *maximum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

We consider several subproblems of MAX-SAT here. For every integer $k \geq 2$, we define the **MAX-kSAT** problem as a subproblem of MAX-SAT, where the problem instances are formulae in kCNF[33]. For every integer $k \geq 2$, we define the **MAX-EkSAT** as a subproblem of MAX-kSAT, where the inputs are formulae consisting of clauses of the size k only. Each clause $l_1 \vee_2 \vee \cdots \vee l_k$ of such a formula is a Boolean function over exactly k variables, i.e., $l_i \neq l_j$ and $l_i \neq \bar{l}_j$ for all $i, j \in \{1, \ldots, k\}$, $i \neq j$.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Linear Programming (LP)

Input: A matrix $A = [a_{ij}]_{i=1,\ldots,m, j=1,\ldots,n}$, a vector $b \in \mathbb{R}^m$, and a vector $c \in \mathbb{R}^n$, $n, m \in \mathbb{N} - \{0\}$.

Constraints: $\mathcal{M}(A, b, c) = \{X \in \mathbb{R}^n \mid A \cdot X = b$ and the elements of X are non-negative reals only$\}$.

Costs: For every $X = (x_1, \ldots, x_n) \in \mathcal{M}(A, b, c)$, $c = (c_1, \ldots, c_n)^{\mathsf{T}}$, $cost(X, (A, b, c)) = c^{\mathsf{T}} \cdot X = \sum_{i=1}^{n} c_i x_i$.

Goal: *minimum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Integer Linear Programming (IP)

Input: An $m \times n$ matrix $A = [a_{ij}]_{i=1,\ldots,m, j=1,\ldots,n}$, and two vectors $b = (b_1, \ldots, b_m)^{\mathsf{T}}$, $c = (c_1, \ldots, c_n)^{\mathsf{T}}$ for some $n, m \in \mathbb{N} - \{0\}$, a_{ij}, b_i, c_j are integers for $i = 1, \ldots, m$, $j = 1, \ldots, n$.

Constraints: $\mathcal{M}(A, b, c) = \{X = (x_1, \ldots, x_n) \in \mathbb{Z}^n \mid AX = b \text{ and } x_i \geq 0 \text{ for } i = 1, \ldots, n\}$.

Costs: For every $X = (x_1, \ldots, x_n) \in \mathcal{M}(A, b, c)$, $cost(X, (A, b, c)) = \sum_{i=1}^{n} c_i x_i$.

Goal: *minimum*.

Note that IP is not a subproblem of LP, because we did not restrict the language of inputs only, but also the constraints.

The **0/1-Linear Programming (0/1-LP)** is the optimization problem with the language of input instances of IP and the additional constraints requiring that $X \in \{0,1\}^n$ (i.e., that $\mathcal{M}(A, b, c) \subseteq \{0,1\}^n$).

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

Maximum Linear Equation Problem Mod k (MAX-LINMODk)

Input: A set S of m linear equations over n unknowns, $n, m \in \mathbb{N} - \{0\}$, with coefficients from $\mathbb{Z}_k$.
(An alternative description of an input is an $m \times n$ matrix over $\mathbb{Z}_k$ and a vector $b \in \mathbb{Z}_k^m$).

Constraints: $\mathcal{M}(S) = \mathbb{Z}_k^m$
{a feasible solution is any assignment of values from $\{0, 1, \ldots, k-1\}$ to the n unknowns (variables)}.

Costs: For every $X \in \mathcal{M}(S)$,
$cost(X, S)$ is the number of linear equations of S satisfied by X.

Goal: *maximum*.

- 它对应的判定问题是什么？

你理解这个优化问题了吗？

For every prime k, and every positive integer m, we define the problem **MAX-EmLINMODk** as the subproblem of MAX-LINMODk, where the input instances are sets of linear equations such that every linear equation has at most m nonzero coefficients (contains at most m unknowns).

- 它对应的判定问题是什么？

你能比较一个优化问题和它对应的判定问题的“难度”吗？

Minimum Vertex Cover Problem (MIN-VCP)

Input: A graph $G = (V, E)$.
Constraints: $\mathcal{M}(G) = \{S \subseteq V \mid$ every edge of E is incident to at least one vertex of $S\}$.
Cost: For every $S \in \mathcal{M}(G)$, $cost(S, G) = |S|$.
Goal: *minimum*.

VERTEX COVER PROBLEM.

The vertex cover problem is to decide, for a given graph G and a positive integer k, whether G contains a vertex cover of cardinality k. Remember that a vertex cover of $G = (V, E)$ is any set S of vertices of G such that each edge from E is incident to at least one vertex in S.

Formally, the **vertex cover problem** (**VCP**) is the decision problem $(\text{VCP}, \{0, 1, \#\})$, where

$$\textbf{VCP} = \{u\#w \in \{0,1,\#\}^+ \mid u \in \{0,1\}^+ \text{ and } w \text{ represents a graph that contains a vertex cover of size } Number(u)\}.$$

OT

- 请调研教材中未介绍过的至少2种判定问题和2种优化问题（不能是相互对应版本，也不能是教材中介绍过的问题的对应版本），讨论适用场景，用教材中的方法形式化描述问题。