算法分析与设计

第9章 NP完全性理论

主讲人：甘文生 PhD

Email: wsgan001@gmail.com

暨南大学网络空间安全学院

Fall 2021

Jinan University, China

内容提要：

- 理解RAM，RASP和图灵机计算模型
- 理解非确定性图灵机的概念
- 理解P类与NP类语言的概念
- 理解NP完全问题的概念
- 理解近似算法的性能比及多项式时间近似格式的概念
- 通过范例学习NP完全问题的近似算法
  - 顶点覆盖问题；
  - 旅行售货员问题；
  - 集合覆盖问题；
  - 子集和问题。

计算机科学的局限性

- 可解性：问题及其可解性可用函数和可计算性来代替
- 可计算性理论：研究计算的一般性质的数学理论，它通过建立计算的数学模型（例如抽象计算机），精确区分哪些是可计算的，哪些是不可计算的。
- 可计算函数：能够在抽象计算机上编出程序计算其值的函数。这样就可以讨论哪些函数是可计算的，哪些函数是不可计算的。
- Church-Turing论题(又称邱奇-图灵猜想，邱奇论题，邱奇猜想，图灵论题)：若一函数在某个合理的计算模型上可计算，则它在图灵机上也是可计算的。
- 不可计算性：很多问题和函数是无法用具有有穷描述的过程完成计算

Polynomial Time (多项式时间)

- 什么是 “多项式时间 (Polynomial Time)”?

Polynomial Time

Non-Polynomial Time

$f(n) \setminus n$	10	10^2	10^3
$log_2 n$	3.3	6.6	10
n	10	10^2	10^3
$nlog_2 n$	0.33×10^2	0.7×10^3	10^4
n^2	10^2	10^4	10^6
2^n	1024	1.3×10^3	$> 10^{100}$
$n!$	3^6	$> 10^{100}$	$> 10^{100}$

Polynomial Time (多项式时间)

Definition:

- ✓ 一个称为多项式时间的算法(**Polynomial-time Algorithm**) 必须符合：在合理的输入大小 (input size)下，该算法在最差情况(Worst-case)的时间复杂度以多项式函数为限。
- ✓ 因此，若n为input size，存在一个多项式函数 $p(n)$，则：

$$W(n) \in O(p(n)).$$

Polynomial-time computable

- ✓ 一个函数f(x) 为polynomial-time computable，并且若存在一个算法，使得对所有的输入 x，皆可在Polynomial Time内求得 f。

Intractability (难解问题)

- 如果我们说「这个问题很难」，可能有两种不同的含义：
  - ✓ [意义 1]: 这个问题也许目前已经有一些还不错的近似解法，只是想进一步找出真正最佳的方法是件困难的事
  - ✓ [意义2]:这个问题本身就难以找出解决方法。
- 第一种意思指的是对**人**而言很困难，而第二种意思指的是对**计算机**而言很难。
- 在探讨问题的难度时，比较正确的讲法应该是指一个问题是**易解的 (tractable)** 或是**难解的 (intractable)**。

Intractability (难解问题)

- 在计算机科学领域，若無法在最差情况(Worst-case)下，以多项式时间的算法來解决某个问题，则该问题被称为难解 (**Intractable**)问题
  - ✓ 一个难解的问题，必须沒有任何多项式时间的算法可以解它。
- 如果有一个问题在最差情况(Worst-case)下，目前还找不到一个Polynomial-Time Algorithm解它，但是也無法保证未來就找不到Polynomial-Time Algorithm來解这个问题，则***无法证明该问题是Intractable***.
  - 早期，利用 **brute-force algorithm (暴力算法)** 解连续矩阵相乘问题 (Chained Matrix Multiplication problem)，其时间复杂度为**non-polynomial time**.
  - 然而，若以 **dynamic programming algorithm** (动态规划) 來解，则其时间复杂度为$\Theta(n^3)$.

Intractability (难解问题)

- 就难解性而言，问题主要可分为三种:
  - ✓ Problems for which polynomial-time algorithms **have been found**
    - 如：最短路径问题、MST问题、排序问题、搜索问题…
  - ✓ Problems that have been proven to be **intractable**
    - 时间复杂度被证明为**指数复杂度(以上)**的问题。如：汉纳塔问题
    - **不存在有解决问题的算法**的问题。如：代码停止问题 (Halting Problem)、功能相等问题(Equivalence Problem)…
  - ✓ Problems that have not been proven to be intractable, but for which polynomial-time algorithms have never been found

大多数的问题不是落在第一类，就是落在第三类。

- ✓ 第三类问题中，常见的有**旅行推销员问题 (Traveling Salesman Problem)**

Intractability (难解问题)

- 若采用暴力法去解TSP问题，则会发现要找出所有可能的路径所花费用的时间呈**指数 (Exponentially)** 增长!!
  - ✓ 3 cities → 1 solution.
  - ✓ 10 cities → 181,440 possible tours
  - ✓ n cities → (n-1)!/2 possible tours
- 若 n=26，则有 25! /2条不同路径：
  - ✓ 25! = 15511210043330985984000000 ≅ 1.55 x 10^{25}这个数字究竟有多大？假设电脑每秒可计算 10^6 条路径的成本，一年有 3.15 x 10^7秒， 故一年可计算 3.15 x 10^{13}条路径，求出所有路径的成本需要 5 x 10^{11}年
  - ✓ 当 n=26，就需要五千亿年，很显然该方法毫无用处。

计算模型

- 在进行问题的计算复杂性分析之前，首先必须建立求解问题所用的计算模型，包括定义该计算模型中所用的基本运算。
- 建立计算模型的目的是为了使问题的计算复杂性分析有一个共同的客观尺度。

3个基本计算模型：

- 随机存取机RAM (Random Access Machine)
- 随机存取存储程序机RASP (Random Access Stored Program Machine)
- 图灵机 (Turing Machine)。

这3个计算模型在计算能力上是等价的，但在计算速度上是不同的。

Deterministic v.s. Non-deterministic

Deterministic Algorithm (决定性算法)

- ✓ **Def: 这类算法在做任何事时，该算法的下一步只有一件事可以做。(Permitting at most one next move at any step in a computation)**
- ✓ **是指算法中每一个步骤的运算都需要被唯一定义，因此产生的结果也是唯一的。**
- ✓ **能夠执行决定性算法的机器，称为决定性的机器 (Deterministic Machine)。电脑就是一种决定性的机器。**
  - **由于此类计算机器运作的算法在处理问题时，每一步只有一件事，因此，只要有一个处理器即可，故容易实现。**

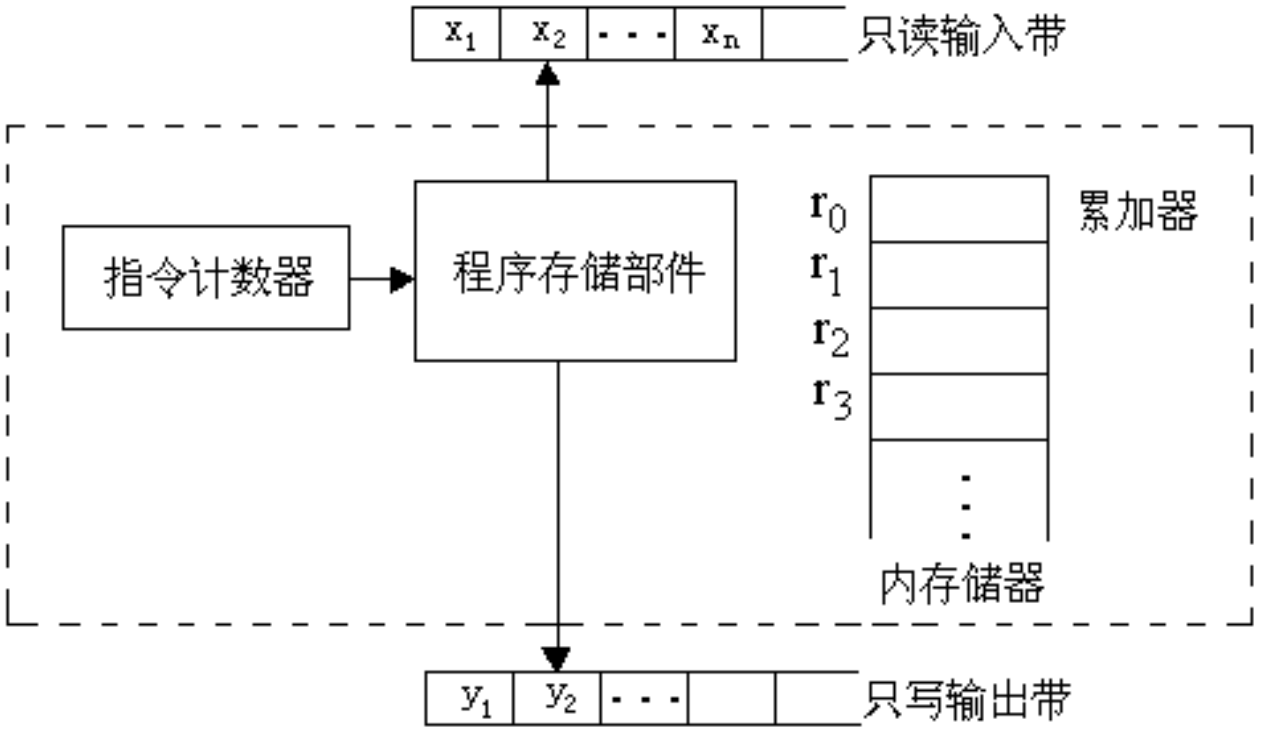
Deterministic v.s. Non-deterministic

Non-deterministic Algorithm (非决定性算法)

- ✓ **Def: 这类算法在做任何事时，该算法的下一步可能会有无限多个件事可以选择。 (Permitting more than one choice of next move at some step in a computation)**
- ✓ **算法中每一个步骤的运算无法被唯一定义。**
- ✓ **能够执行非决定性算法的机器，称为非决定性的机器 (Non-deterministic Machine)。**
  - **由于非决定性算法在执行时，每一步可能有无限多个件事要处理，故非决定性计算机前需假设有无限多个处理器可平行处理處理。因此，非决定性计算机器的计算能力比决定性计算机前要強大。**
  - **但是，实际上并不存在此种机器。**

随机存取机RAM

1、RAM的结构



随机存取机RAM

2、RAM程序

一个RAM程序定义了从输入带到输出带的一个映射。可以对这种映射关系作2种不同的解释。

解释一：把RAM程序看成是计算一个函数。若一个RAM程序P总是从输入带前n个方格中读入n个整数x1，x2，…，xn，并且在输出带的第一个方格上输出一个整数y后停机，那么就说程序P计算了函数f(x1，x2，…，xn) = y

解释二：把RAM程序当作一个语言接受器。将字符串S = a1a2…an放在输入带上。在输入带的第一个方格中放入符号a1，第二个方格中放入符号a2，…，第n个方格中放入符号an。然后在第n+1个方格中放入0，作为输入串的结束标志符。如果一个RAM程序P读了字符串S及结束标志符0后，在输出带的第一格输出一个1并停机，就说程序P接受字符串S。

随机存取机RAM

3、 RAM程序的耗费标准

标准一：均匀耗费标准。在均匀耗费标准下，每条RAM指令需要一个单位时间；每个寄存器占用一个单位空间。以后除特别注明，RAM程序的复杂性将按照均匀耗费标准衡量。

标准二：对数耗费标准。对数耗费标准是基于这样的假定，即执行一条指令的耗费与以二进制表示的指令的操作数长度成比例。在RAM计算模型下，假定一个寄存器可存放一个任意大小的整数。因此若设l(i)是整数i所占的二进制位数，则

$$l(i)=\begin{cases}\lfloor \log|i| \rfloor & i\neq 0\\ 1 & i=0\end{cases}$$

随机存取存储程序机RASP

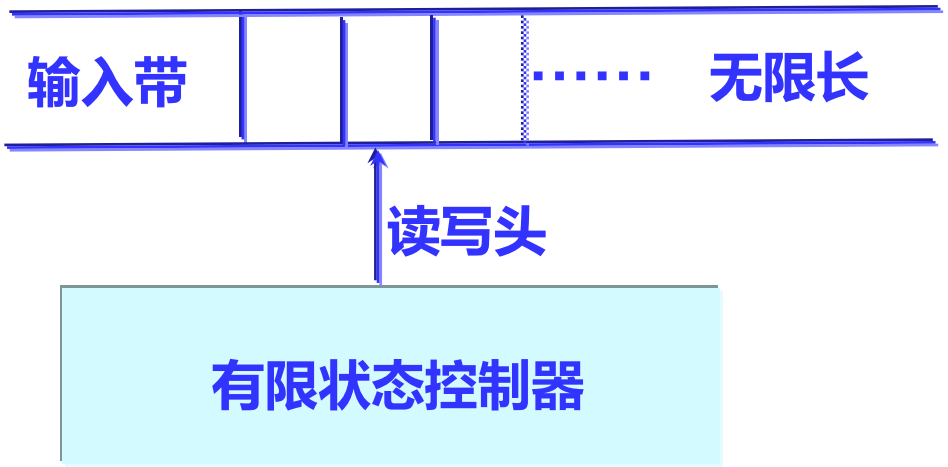
1、RASP的结构

RASP的整体结构类似于RAM，所不同的是RASP的程序是存储在寄存器中的。每条RASP指令占据2个连续的寄存器。第一个寄存器存放操作码的编码，第二个寄存器存放地址. RASP指令用整数进行编码。

2、RASP程序的复杂性

不管是在均匀耗费标准下，还是在对数耗费标准下，RAM程序和RASP程序的复杂性只差一个常数因子。在一个计算模型下T(n)时间内完成的输入-输出映射可在另一个计算模型下模拟，并在kT(n)时间内完成。其中k是一个常数因子。空间复杂性的情况也是类似的。

图灵机



有单带、多带等变种，
计算能力等价

依据控制器的状态和读写头所读字符，决定执行以下3个操作之一、之二或全部：

1）改变有限状态控制器中的状态；

2）读写头在相应的方格中写一符号；

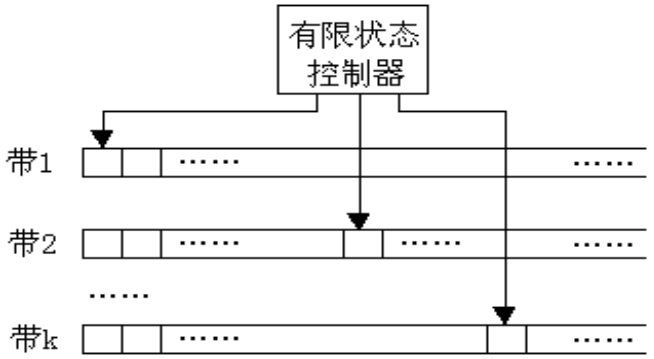
3）读写头左、右移一格或不动。

确定型图灵机DTM：若对任一个状态和符号，要执行的动作都是唯一的

非确定型图灵机NTM：若执行的动作是有穷多个可供选择

图灵机

- 根据有限状态控制器的当前状态及每个读写头读到的带符号，图灵机的一个计算步可实现下面3个操作之一或全部。(1)改变有限状态控制器中的状态。(2)清除当前读写头下的方格中原有带符号并写上新的带符号。(3)独立地将任何一个或所有读写头，向左移动一个方格(L)或向右移动一个方格(R)或停在当前单元不动(S)。



多带图灵机

- k带图灵机可形式化地描述为一个7元组

(Q，T，I，δ，b，q0，qf)，其中:

(1)Q是有限个状态的集合。(2)T是有限个带符号的集合。

(3)I是输入符号的集合，I T。(4)b是唯一的空白符，b∈T-I。

(5)q0是初始状态。　　(6)qf是终止(或接受)状态。

(7)δ是移动函数。它是从QTk的某一子集映射到Q (T{L，R，S})k的函数。

图灵机

- 与RAM模型类似，图灵机既可作为语言接受器，也可作为计算函数的装置。

- 图灵机M的时间复杂性T(n)是它处理所有长度为n的输入所需的最大计算步数。如果对某个长度为n的输入，图灵机不停机，T(n)对这个n值无定义。

- 图灵机的空间复杂性S(n)是它处理所有长度为n的输入时，在k条带上所使用过的方格数的总和。如果某个读写头无限地向右移动而不停机，S(n)也无定义。

P类与NP类问题

- 一般地说，将可由多项式时间算法求解的问题看作是易处理的问题，而将需要超多项式时间才能求解的问题看作是难处理的问题。
- 有许多问题，从表面上看似乎并不比排序或图的搜索等问题更困难，然而至今人们还没有找到解决这些问题的多项式时间算法，也没有人能够证明这些问题需要超多项式时间下界。
- 在图灵机计算模型下，这类问题的计算复杂性至今未知。
- 为了研究这类问题的计算复杂性，人们提出了另一个能力更强的计算模型，即**非确定性图灵机计算模型**，简记为NDTM (Nondeterministic Turing Machine)。
- 在非确定性图灵机计算模型下，许多问题可以在多项式时间内求解。

非确定性图灵机

- 在图灵机计算模型中，移动函数δ是单值的，即对于$Q \times T^k$中的每一个值，当它属于δ的定义域时，$Q\times(T\times\{L，R，S\})^k$中只有唯一的值与之对应，称这种图灵机为**确定性图灵机**，简记为**DTM (Deterministic Turing Machine)**。

- **非确定性图灵机(NDTM)**：一个k带的非确定性图灵机M是一个7元组：$(Q，T，I，\delta，b，q_0，q_f)$。与确定性图灵机不同的是非确定性图灵机允许移动函数δ具有不确定性，即对于$Q\times T^k$中的每一个值$(q; x1, x2, \ldots, x_k)$，当它属于δ的定义域时，$Q\times(T\times\{L，R，S\})^k$中有唯一的一个子集$\delta(q; x1, x2, \ldots, x_k)$与之对应。可以在$\delta(q; x1, x2, \ldots, x_k)$中随意选定一个值作为它的函数值。

P类与NP类语言

- P类和NP类语言的定义：

  P = {L|L是一个能在多项式时间内被一台DTM所接受的语言}

  NP = {L|L是一个能在多项式时间内被一台NDTM所接受的语言}

由于一台确定性图灵机可看作是非确定性图灵机的特例，所以可在多项式时间内被确定性图灵机接受的语言也可在多项式时间内被非确定性图灵机接受。故$P \subseteq NP$。

P、NP及NPC类问题

- **P类问题**：一类问题的集合，对其中的任一问题，都存在一个确定型图灵机M和一个多项式p，对于该问题的任何(编码)长度为n的实例，M都能在p(n)步内，给出对该实例的回答。即：**多项式时间内可被解决的问题**
- **NP类问题**：一类问题的集合，对其中的任一问题，都存在一个非确定型图灵机M和一个多项式p，对于该问题的任何(编码) 长度为n的实例，M都能在p(n)步内，给出对该实例的回答。

若NTM在每一步都恰有2步可供选择，则回答实例需考察2p(n)种不同的可能性。

存在多项式时间的算法吗？

多项式时间内可验证问题(指验证其解的正确性)

P、NP及NPC类问题

- **多一归约**：假设L_1和L_2是两个判定问题，f将L1的每个实例I变换成L_2的实例f(I)。若对L_1的每个实例I，I的答案为“是”当且仅当f(I)是L_2的答案为“是”的实例，则称f是从L_1到L_2的多一归约，记作$L_1 \leq_m L_2$(传递关系)

  直观意义：将求解L_1的问题转换为求解L_2的问题，而问题L_1不会难于L_2

- **多项式时间多一归约**：若f是多项式时间可计算，则上述归约称为多项式时间多一归约，也称多项式时间变换。记作：$L_1 \leq_m^p L_2$

- **NPC问题**：对于一个(判定性)问题q，若 (1) $q \in NP$; (2) NP中任一问题均可多项式时间多一归约到q，则称**问题q为NP-完全的(NP-complete，NPC)**

- **NP-hard问题**：若问题q仅满足条件(2)而不一定满足条件(1)，则**问题q称为NP-难的(NP-hard，NPH)**。显然：NPC $\subseteq$ NP-hard

- **NPC和NP-hard关系**：

  NP-hard问题至少跟NPC问题一样难。

  NPC问题肯定是NP-hard的，但反之不一定

P, NP, NP hard, NP Complete

- **P问题:**
  - 是一组Decision Problem的集合，这些问题皆可利用Deterministic Algorithm在Worst Case的情况下，在**Polynomial Time**的复杂度内**被解决**。
- **NP问题:**
  - 是指Non-deterministic和Polynomial两个单词的简写
  - 这类问题只要给出一个解答，可以在Polynomial Time的复杂度内很快地**验证 (Verify)** 出这个解答是否正确。
    - 由于决定性算法为非决定性算法的一个特例，且容易找到答案也会容易验证答案。因此，P可视为NP的一个特例。
    - P⊆NP (✓)
    - P=NP (?)

P, NP, NP hard, NP Complete

- **NP-hard:**
  - 是一组Decision Problem的集合。当 Q ∈ NP-hard 且唯若所有属于NP的Decision Problem Q' (∀Q'∈NP) 皆可**多项式时间转化**成Q (Q'∝Q)
  - 此类问题至今仍未找到一个多项式复杂度的决定性算法，且一般相信没有多项式复杂度的决定性算法存在。
- **NP-complete:**
  - 是一组Decision Problem的集合。若某一个问题Q属于NP-complete，则满足以下两个条件:
    - Q属于**NP**
    - Q属于**NP-hard**

P、NP及NPC类问题

- **NP =？ P**

∵确定型图灵机是非确定型图灵机的特例，∴P⊆NP

是否有NP⊆P？即是否NP=P？

美国麻省的Clay数学研究所于2000年5月24日在巴黎法兰西学院宣布：对七个“千年数学难题”中的每一个均悬赏100万美元，而问题NP =？ P位列其首：

1.P问题对NP问题

2.霍奇猜想

3.庞加莱猜想(2002.11-2003.7，俄罗斯数学家佩雷尔曼在3篇论文预印本中证明了几何化猜想，2006被授予菲尔兹奖)

4.黎曼假设

5.杨－米尔斯存在性和质量缺口

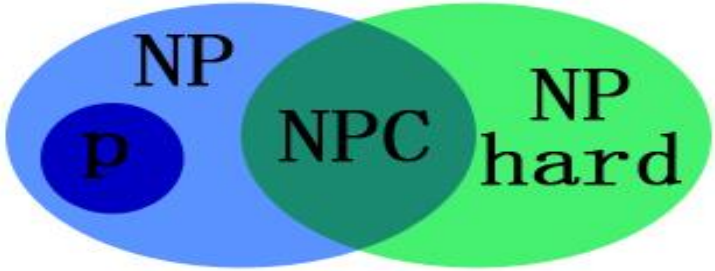
6.纳维叶－斯托克斯方程的存在性与光滑性

7.贝赫和斯维纳通－戴尔猜想

P、NP及NPC类问题

- **P、NP、NPC和NP-hard之关系**

  NPC是NP中最难的问题，但是NP-hard至少与NPC一样难



- **如何证明问题q是NP-hard或是NPC的？**

若已知q' ∈ NPC或q' ∈ NPH，且q' ≤ pq，则q ∈ NPH；若进一步有q ∈ NP，则q ∈ NPC。

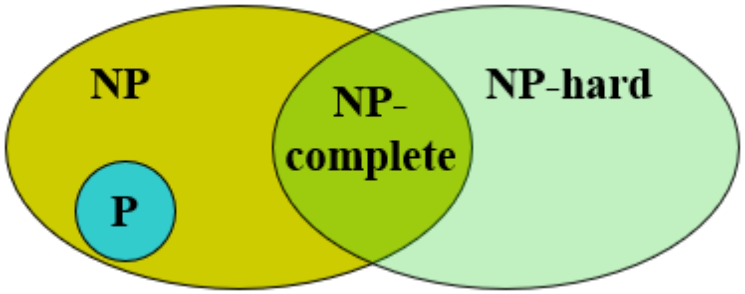
即：要证q是NPH的，只要找到1个已知的NPC或NPH问题q'，然后将q'多项式归约到q即可。若还能验证q ∈NP，则q是NPC的。

∵ NP中任意问题均可多项式归约到q'，由于≤p有传递性

∴ 它们也都能多项式归约到q，由定义可知q是NPH的

P、NP及NPC类问题

- P、NP、NPC和NP-hard的关系



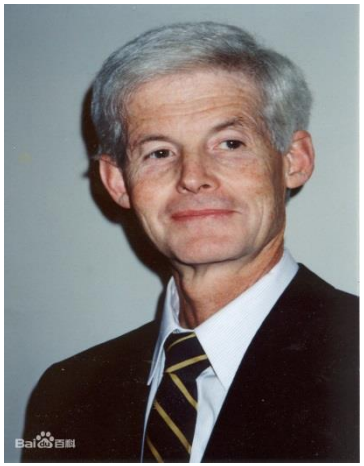
- NP-hard与NP-complete的关系：
  - 所有NP-complete问题都是NP-hard问题 (如：旅行推销员问题)；但是NP-hard问题不一定是NP-complete问题(如：程式停止问题，它不是NP问题)
  - NP-complete ⊆ NP-hard
  - 如果有一个NP-hard问题能找到多项式复杂度的决定性算法，则所有NP-complete问题也都存在多项式复杂度的决定性算法。

Summary

- Nearly all of the decision problems are NP problems.
- In NP problems, there are some problems which have polynomial algorithms. They are called P problems.
  - Every P problem must be an NP problem.
- There are a large set of problems which, up to now, have no polynomial algorithms.

NP-完全性理论

- **Cook的贡献：第一个NPC问题**



史提芬·库克(Stephen Arthur Cook，1939－)NP完全性理论的奠基人，在其1971年论文"The Complexity of Theorem Proving Procedures"中，给出了第一个NP完备的证明，即Cook定理：可满足性问题(Satisfiability problem) 是NP完全问题，亦即 SAT∈NPC。且证明了：SAT∈P当且仅当P = NP

Cook于1961年获Michigan大学学士学位，1962和1966年分获哈佛大学硕士与博士学位。1966-1970，他在UC Berkeley担任助教授；1970年加盟多伦多大学，现为该校CS和数学系教授，他开启了NP完备性的研究，令该领域于之后的十年成为计算机科学中最活跃和重要的研究。因其在计算复杂性理论方面的贡献，尤其是在奠定NP完全性理论基础上的突出贡献而荣获1982年度的图灵奖。

NP-完全性理论

Karp的贡献

理查德·卡普(Richard Karp，1935-) 1972年论文"Reducibility among Combinatorial Problems"发展和加强了由库克提出的"NP完全性"理论。尤其是库克仅证明了命题演算的可满足问题是NP完全的，而卡普则证明了从组合优化中引出的大多数经典问题(背包问题、覆盖问题、匹配问题、分区问题、路径问题、调度问题等)都是NP完全问题。只要证明其中任一个问题是属于P类的，就可解决计算复杂性理论中最大的一个难题，即P =? NP。

Karp于1955、1956和1959年分别获哈佛大学文学学士、理学硕士和应用数学博士学位，现任UC Berkeley计算机科学讲座教授，美国科学院、美国工程院、美国艺术与科学院、欧洲科学院院士。因其在计算机科学领域的基础贡献曾获图灵奖(1985)、冯诺依曼奖、美国国家科学勋章、哈佛大学百年奖章等奖项.

NP-完全性理论

NP-完全性理论的局限性

易解问题：可多项式时间内求解的问题

难解问题：需超多项式时间求解的问题

NP-完全性理论既没有找到第二类问题的多项式时间的算法，也没有证明这样的算法就不存在，而是证明了这类问题计算难度之等价性（彼此间困难程度相当）。因此，NPC具有如下性质：若其中1个问题多项式可解当且仅当其他所有NPC问题亦多项式可解

难解问题与易解问题之相似性

1) 最短/最长简单路径

单源最短路径问题：对有向图G，时间O(VE)，P问题

两点间最长路径：NPC问题，即使所有边上权为1

2) 欧拉环/哈密尔顿圈 (G为无向图或有向图)

欧拉环：G中有通过每条边恰好一次的环？P，多项式时间可解

哈氏圈：G中有通过每个顶恰好1次的圈？NPC

NP完全问题的求解

- 减少搜索量
  - 简单算法是穷举搜索，时间为指数
  - 减少搜索量：分枝限界法，隐枚举法、动态规划等。可以提高效率，但时间复杂度不变
- 优化问题
  - 降低优化要求，求近似解，以得到一个多项式时间的算法。即：找寻在容许的时间内得到容许精度的近似最优解的算法

P类与NP类语言

- NP类语言举例——无向图的团问题。

- 该问题的输入是一个有n个顶点的无向图G = (V，E)和一个整数k。要求判定图G是否包含一个k顶点的完全子图(团)，即判定是否存在V'⊆V，|V'|=k，且对于所有的u, v ∈ V'，有(u, v) ∈ E。
- 若用邻接矩阵表示图G，用二进制串表示整数k，则团问题的一个实例可以用长度为 $n^2+\log k+1$ 的二进位串表示。

因此，团问题可表示为语言：

CLIQUE={w#v | w，v∈{0，1}*，以w为邻接矩阵的图G有一个k顶点的团，其中v是k的二进制表示}

P类与NP类语言

- 接受该语言CLIQUE的非确定性算法：用非确定性选择指令选出包含k个顶点的候选顶点子集V，然后确定性地检查该子集是否是团问题的一个解。算法分为3个阶段：
- 算法的第一阶段将输入串w#v分解，并计算出$n = \sqrt{|w|}$，以及用v表示的整数k。若输入不具有形式w#v或|w|不是一个平方数就拒绝该输入。显而易见，第一阶段可 $O(n^2)$ 在时间内完成。
- 在算法的第二阶段中，非确定性地选择V的一个k元子集V'⊆V。
- 算法的第三阶段是确定性地检查V'的团性质。若V'是一个团则接受输入，否则拒绝输入。这显然可以在$O(n^4)$ 时间内完成。因此，整个算法的时间复杂性为 $O(n^4)$ 。

非确定性算法在多项式时间内接受语言CLIQUE，故CLIQUE∈NP。

多项式时间验证

- 多项式时间可验证语言类VP可定义为：VP = {L | L∈∑*，∑为一有限字符集，存在一个多项式p和一个多项式时间验证算法A(X, Y)使得对任意X∈∑*，X∈L当且仅当存在Y∈∑*，|Y| ≤ p(|X|)且A(X, Y) = 1}。

- 定理：VP = NP。

例如(哈密顿回路问题)：一个无向图G含有哈密顿回路吗?

无向图G的哈密顿回路是通过G的每个顶点恰好一次的简单回路。可用语言HAM-CYCLE 定义该问题如下：

HAM-CYCLE = {G|G含有哈密顿回路}

NP完全问题

- $P \subseteq NP$。
- 直观上看，P类问题是确定性计算模型下的易解问题类，而NP类问题是非确定性计算模型下的易验证问题类。
- 大多数的计算机科学家认为NP类中包含了不属于P类的语言，即$P \neq NP$。
- NP完全问题有一种令人惊奇的性质，即如果一个NP完全问题能在多项式时间内得到解决，那么NP中的每一个问题都可以在多项式时间内求解，即P=NP。
  - ✓ 目前还没有一个NP完全问题有多项式时间算法。

多项式时间变换

设$L_1 \subseteq \Sigma_1^*$，$L_2 \subseteq \Sigma_2^*$是2个语言。所谓语言L_1能在多项式时间内变换为语言（简记为$L_2 \propto_p L_1$）是指存在映身f: L_2，且f满足：$\Sigma_1^* \rightarrow \Sigma_2^*$

(1)有一个计算f的多项式时间确定性图灵机；

(2)对于所有x$\in \Sigma_1^*$，x$\in L_1$，当且仅当f(x)$\in L_2$。

- 定义：语言L是NP完全的当且仅当: (1) $L \in$ NP；(2) 对于所有$L' \in$ NP有$L' \propto_p L$。

如果有一个语言L满足上述性质(2)，但不一定满足性质(1)，则称该语言是NP难的。所有NP完全语言构成的语言类称为NP完全语言类，记为NPC。

多项式时间变换

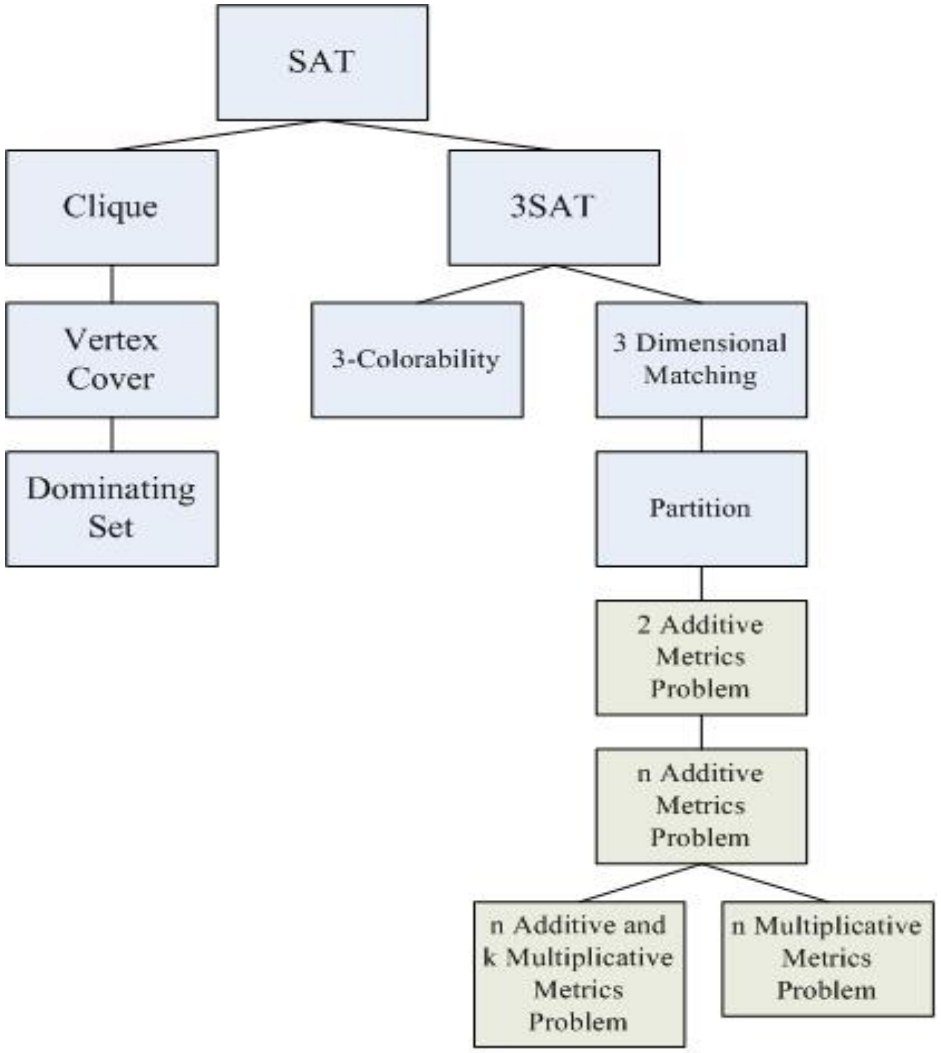
- 定理2：设L是NP完全的，则

(1) $L \in \mathrm{P}$当且仅当$\mathrm{P} = \mathrm{NP}$；

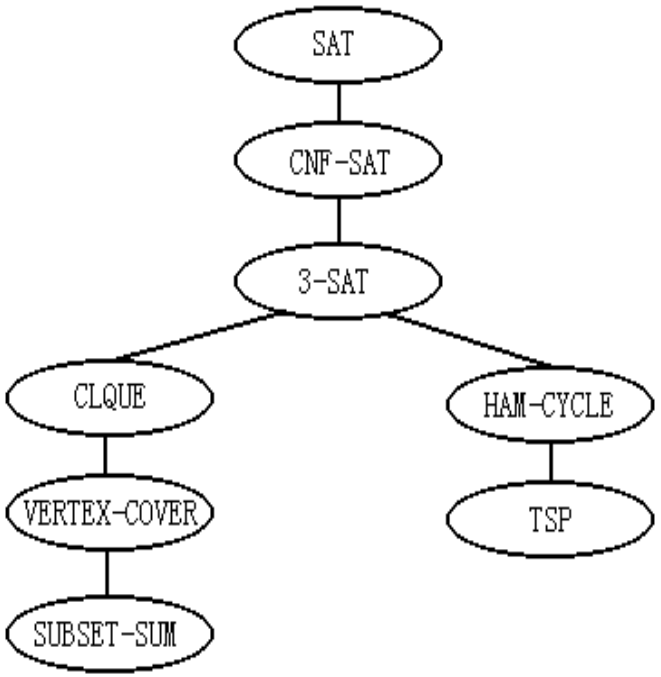
(2) 若$L \propto_{\mathrm{p}} L_1$，且$L_1 \in \mathrm{NP}$，则$L_1$是NP完全的。

定理的(2)可用来证明问题的NP完全性。但前提是：要有第一个NP完全问题L。

一些典型的NP完全问题



Excerpt from "Introduction to Algorithms, UDI MANBER"
Figure 11.1



部分NP完全问题树

一些典型的NP完全问题

- 证明以下的定理：
  - 3-SAT问题为NP-Complete
  - Clique (社区)问题是NP-Complete
  - Vertex-Cover (顶点覆盖)问题为NP-Complete
  - Dominating Set (支配集) 问题为NP-Complete

NP完全问题的近似算法

- 迄今为止，所有的NP完全问题都还没有多项式时间算法。

- 对于这类问题，通常可采取以下几种解题策略。

  (1)只对问题的特殊实例求解

  (2)用动态规划法或分支限界法求解

  (3)用概率算法求解

  (4)只求近似解

  (5)用启发式方法求解

本小节主要讨论解NP完全问题的近似算法。理解近似算法的性能比的概念；理解多项式时间近似格式的概念；通过范例学习NP完全问题的近似算法。

近似算法的概述

- 现实中许多优化问题是NP-hard的，由复杂性理论知：若P≠NP(很可能为真)，就不可能找到多项式时间的算法来对问题的所有输入实例求出最优解。但若放松要求，就可能存在有效求解算法。

实用中可考虑3种放宽要求的可能性：

- **1. 超多项式时间启发**

不再要求多项式时间算法，有时求解问题存在超多项式时间算法，实用中相当快。例如，0/1背包问题是NPC问题，但存在1个伪多项式时间算法很容易解决它.

缺点：该技术只对少数问题有效。

近似算法的概述

2. 启发式概率分析

不再要求问题的解满足所有的输入实例。在某些应用中，有可能输入实例的类被严格限制，对这些受限实例易于找到其有效算法。而这种结果往往归功于对输入实例约束的概率模型。

缺点：选取一个特殊的输入分布往往是不易的。

3. 近似算法

不再要求总是找到最优解。在实际应用中有时很难确定一个最优解和近似最优解(次优解)之间的差别，因问题的输入实例数据本身就可能是近似的。

设计一个算法能够求出所有情况下的次优解来解NP-hard问题往往是真正有效的手段。

近似算法的概述

近似算法设计思想

- 放弃求解最优解，用近似最优解代替最优解，以此换取：
  - 算法设计上的简化
  - 时间复杂性的降低
- 近似算法是可行的：
  - 问题的输入数据是近似的；
  - 问题的解允许有一定程度的误差；
  - 近似算法可在很短的时间内得到问题的近似解。

近似算法的概述

- 优化问题近似解分类

1）容易近似

Knapsack，Scheduling，Bin Packing等

2）中等难度

Vertex Cover，Euclidean TSP，Steiner Trees等

3）难于近似

Graph Coloring，TSP，Clique等

这类问题即使找到很差的近似解也是NP-hard

近似算法的性能

- 衡量近似算法性能的标准：
  - ✓ 时间复杂性：必须是多项式阶的——基本目标
  - ✓ 解的近似程度：——重要目标
- 若一个最优化问题的最优值为c*，求解该问题的一个近似算法求得的近似最优值为c，则将该近似算法的近似比定义为

最小化问题 $c \geq c^*$

$$\eta = \max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\}$$

最大化问题，$c^* \geq c$

$\eta \geq 1$；且η越大，近似解越差！

- 在通常情况下，该性能比是问题输入规模n的一个函数 ρ (n)，即

$$\max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\} \leqslant \rho(n)$$

近似算法总结

- 近似算法放弃求最优解，用近似解代替最优解，以换取算法设计上的简化和时间复杂性的降低。
- 近似算法通常采用两个标准来衡量性能：
  - ✓ 算法的时间复杂性
  - ✓ 解的近似程度
    - 近似比η
    - 相对误差λ
    - 相对误差界ε(n)

谢谢！

Q & A