

# 算法设计与分析

# 第10章 NP完全问题

学习要点:

- ◆ 确定算法和不确定算法
- ◆ 判定问题和最优化问题的关系
- ◆ 可满足性问题
- ◆ P类问题和NP类问题
- ◆ NP难度(NP hard)和NP完全(NP complete)问题
- ◆ Cook定理
- ◆ 典型的NP完全（或NP难度）问题的证明



- 章节内容:

- 10.1 基本概念

- 10.2.1 Cook定理

- 10.3 一些典型的NP完全问题



# 10.1 基本概念

- ◆ 将能在多项式时间内求解的问题视为易处理问题(tractable problem)。
- ◆ 至今尚未找到多项式时间算法求解的问题视为难处理问题(intractable problem)。
  - NP 完全问题或 NP 难度(NP hard)问题
  - 如：指数时间算法

⊕ 如果任意一个 NP 难度问题存在一个多项式时间算法，那么所有 NP 完全问题都可以在多项式时间内求解。

⊕ 一个 NP 完全问题可以在多项式时间内求解，当且仅当所有其他的 NP 完全问题都可以在多项式时间内求解。

# 10.1.1 不确定算法和不确定机

**不确定算法**的抽象计算模型：

- 算法在抽象机上运行与计算机系统的性能无关；
- 算法的执行表现为执行一个基本运算序列；
- 基本运算的**执行时间**是有限常量；

✓ **Choice(S)**: 任意选择集合S的一个元素。

✓ **Failure()**: 发出不成功完成信号后算法终止。

✓ **Success()**: 发出成功完成信号后算法终止。

包含不确定选择语句，并能按上述方式执行一个算法的机器称为**不确定机 (non deterministic machine)**。

在不确定机上执行的算法称为**不确定算法(non deterministic algorithm)**。

```
{  
  int j=Choice(0,n-1);  
  if(a[j]==x)  
  {   cout<<j;  
      Success();  
  }  
  cout<<-1;  
  Failure();  
}
```

**Choice**函数每一次总能在 **$O(1)$** 时间内做出导致成功的正确选择。

执行时间都为 **$O(1)$**

//不确定算法成功终止

若算法执行中需作出一系列的**Choice**函数选择，当且仅当**Choice**的任何一组选择都不会导致成功信号时，算法在 **$O(1)$** 时间不成功终止。

不确定机的执行方式，可理解为不受限制的并行计算：

- 不确定机执行不确定算法时，每当Choice函数进行选择时，就好像复制了多个程序副本，每一种可能的选择产生一个副本，所有副本同时执行。一旦一个副本成功完成，将立即终止所有其他副本的计算。
- 如果存在至少一种成功完成的选择，一台不确定机总能做出最佳选择，以最短的程序步数完成计算，并成功终止。
- 不确定机能及时判断算法的某次执行不存在任何导致成功完成的选择，并使算法在一个单位时间内输出“不成功”信息后终止。

显然，这种机器是虚构的，是一种概念性计算模型！

## 定义10-1 (不确定算法时间复杂度)

一个不确定算法所需的时间是指对任意一个输入，当存在一个选择序列导致成功完成时，达到成功完成所需的最少程序步。在不可能成功完成的情况下，所需时间总是 $O(1)$ 。

不确定搜索算法：

```
void Search(int a[], T x)
{
    int j = Choice(0, n-1);
    if (a[j] == x)
    {
        cout << j;
        Success();
    }
    cout << -1;
    Failure();
}
```

若元素 $x$ 在数组中，**Choice**函数总能在 $O(1)$ 时间内选中该元素下标，并成功终止。

否则，算法在 $O(1)$ 时间失败终止。因此该不确定搜索算法的时间复杂度为 $O(1)$ 。



## 例10-2 将n个元素的序列排成有序序列。

不确定排序算法：

```
void NSort(int a[],int n)
```

```
{
```

```
    int b[mSize],i,j;
```

```
    for (i=0;i<n;i++) b[i]=0;    //将b初始化为0
```

```
    for (i=0;i<n;i++)
```

```
    {    j=Choice(0,n-1);
```

```
        if (b[j]) Failure();
```

```
        b[j]=a[i];
```

```
    }
```

```
    for (i=0;i<n-1;i++)
```

```
        if (b[i]>b[i+1]) Failure();    //若存在两元素逆序，则失败
```

```
    Success();    //Choice函数的n次正确选择，算法成功
```

```
}
```

必定存在对b中下标的n次恰当选择，使得能将每个a[i]恰好保存在一个空闲的b[j]处，并且进一步确保b中元素排成有序序列，从而能顺利通过随后的有序性验证，导致成功终止。

因此该不确定搜索算法的时间复杂度为O(n)。

# 判定问题和最优化问题

一个只要求产生“0”或“1”作为输出的问题称为**判定问题 (decision problem)**。

许多**最优化问题**都可以得到与其相对应的**判定问题**，且两者往往存在计算上的相关性：

一个**判定问题**能够在多项式时间内求解，**当且仅当**它相应的**最优化问题**可以在多项式时间内求解。

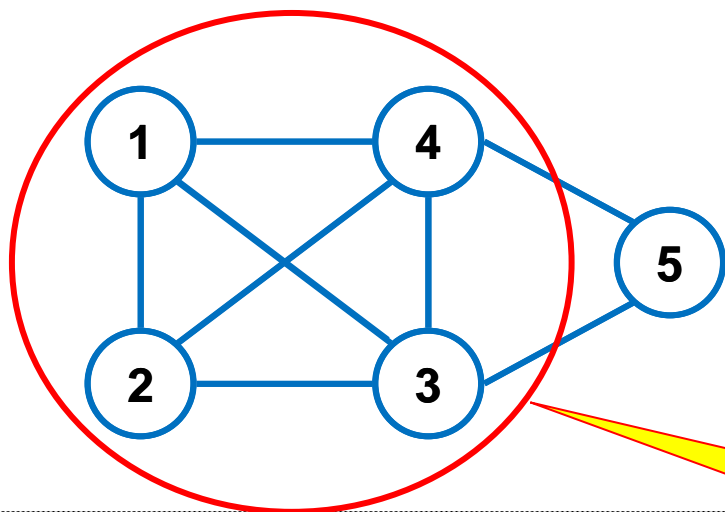
如果**判定问题**不能在多项式时间内求解，那么它相应的**最优化问题**也不能在多项式时间内求解。

### 例10-3 最大集团及其判定问题

无向图 $G=(V,E)$ 的一个完全子图称为该图的一个**集团**，**集团的规模**用集团的顶点数衡量。

➤ **最大集团问题**：确定图 $G$ 的**最大集团规模**的问题。

➤ **最大集团判定问题**：判定图 $G$ 是否存在一个规模至少为 $k$ 的集团。（ $k$ 为给定正整数）



$$S=\{1,2\}, k=|S|=2$$

$$S=\{3,4,5\}, k=|S|=3$$

$$S=\{1,2,3,4\}, k=|S|=4$$

是最大集团

若最大集团问题能在多项式时间 $O(g(n))$ 内求解。

当且仅当

对应的判定问题能在多项式时间 $O(f(n))$ 内求解。

一方面：只需以 $k=1,2,\dots,n$ ，最多 $n$ 次调用最大集团判定算法，便可求得最大集团的大小，因此 $O(g(n))=O(nf(n))$ ；

另一方面：可使用求解最大集团问题的算法，求得最大集团的规模为 $k'$ 。若 $k' \geq k$ ，则最大集团判定问题的解为“1”，否则为“0”。显然有 $O(f(n))=O(g(n))$ 。

许多抽象问题并不是判定问题，而是最优化问题，必须最大化或最小化某个量。然而，如我们看到的，将最优化问题转化为一个并不更难的判定问题通常是比较简单的。

## 10.1.2 可满足性问题

- 数理逻辑中，一个变量  $x_i$  和它的非  $\overline{x_i}$  都称为文字。
- 命题公式是由文字及逻辑运算符“与( $\wedge$ )”和“或( $\vee$ )”构成的表达式。

●如果一个公式具有逻辑与形式： $C_1 \wedge C_2 \wedge \dots \wedge C_k$ ，其中每个子句  $C_i$  都是逻辑或形式  $l_{i1} \vee l_{i2} \vee \dots \vee l_{ip}$ ，每个  $l_{ij}$  是文字，则将这种形式的公式称为合取范式 (conjunctive normal form, **CNF**)。

●如果一个公式具有逻辑或形式： $C_1 \vee C_2 \vee \dots \vee C_k$ ，其中每个子句  $C_i$  都是逻辑与形式  $l_{i1} \wedge l_{i2} \wedge \dots \wedge l_{iq}$ ，每个  $l_{ij}$  是文字，则将这种形式的公式称为析取范式 (disjunctive normal form, **DNF**)。

## 例10-4 可满足性问题 (satisfiability problem)

是一个判定问题，确定对于一个给定的命题公式，是否存在布尔变量的一种赋值（也称真值指派）使该公式为真。

例如：

公式  $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_2})$

是可满足的。只需令  $x_1=1$ ,  $x_2=0$ ,  $x_3=0$ 。

公式  $(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3})$

是不可满足的。



## 程序10-4 可满足性问题的不确定算法

```
void Eval(CNF E, int n)
{
    int x[mSize];
    for (int i=1;i<=n;i++) //O(n)
        x[i]=Choice(0,1); //为变量x[i]赋0或1值
    if (E(x,n)) Success(); //O(e), 计算公式E(x,n)的值
                        //若为真, 成功终止
    else Failure();
}
```

因为：对 $n$ 个布尔变量赋值需要 $O(n)$ 时间，计算公式 $E(x,n)$ 的时间为 $O(e)$ ， $e$ 是公式长度。

所以，可满足性问题的不确定算法时间为 $O(n+e)$ 。



## 10.1.3 P类和NP类问题

**P类问题**：可在多项式时间内用**确定算法求解**的判定问题。

**NP类问题**：可在多项式时间内用**不确定算法求解**的判定问题。（多项式时间内**可验证**问题的解。）

**确定算法是不确定算法**当Choice函数只有一种选择时的特例，所以有：

$$P \subseteq NP$$

但至今无法断定：是否 $P=NP$ 或者 $P \neq NP$ 。



### 定义10-3 多项式约化

令 $Q_1$ 和 $Q_2$ 是两个问题，如果存在一个确定算法A求解 $Q_1$ ，而算法A以多项式时间调用另一个求解 $Q_2$ 的确定算法B。若不计B的工作量，算法A是多项式时间的，则称 $Q_1$ 约化(reduced to)为 $Q_2$ ，记作 $Q_1 \propto Q_2$ 。

即：

- ✓ 求解 $Q_1$ 的确定算法是通过调用求解 $Q_2$ 的确定算法完成的，
- ✓ 对 $Q_2$ 算法实施的调用过程所需的时间是多项式时间的。

那么：只要对问题 $Q_2$ 存在多项式时间求解算法，问题 $Q_1$ 就能在多项式时间内得以求解。

约化存在以下性质：

### 性质10-1

若  $Q_1 \in P$ ,  $Q_2 \propto Q_1$ , 则有  $Q_2 \in P$ 。

### 性质10-2（传递性）

若  $Q_1 \propto Q_2$ ,  $Q_2 \propto Q_3$ , 则  $Q_1 \propto Q_3$ 。



## 10.1.4 NP难度和NP完全问题

### 性质10-4 NP难度 (NP hard)

对任意问题  $Q_1 \in \text{NP}$  都有  $Q_1 \propto Q$ ，则称问题  $Q$  是NP难度 (NP hard) 的。

只要对任何一个NP难度问题  $Q$  找到了它的多项式时间算法，那么，可以断定所有NP类问题都能在多项式时间内求解，因为所有NP类问题都能约化到问题  $Q$ 。

(然而目前尚无任何一个NP难度问题具有多项式时间算法。)

## 性质10-5 NP完全 (NP complete)

对于问题  $Q \in \text{NP}$  且  $Q$  是NP难度的，则称  $Q$  是NP完全 (NP complete, NPC) 的。

所有NP完全问题都是NP难度的，反之不然，NP难度问题不一定是NP完全的（若不是NP类问题，则不是NP完全的）。



现实意义：

若一个问题被证明是**NP难度 (NP hard)** 的，则很难找到一个多项式时间的有效算法。若问题的实例规模较大，则应选择采用**启发式算法**、随机算法或近似算法等其他算法策略求解。

如何确定某个问题是否是**NP难度**的？



证明某个问题Q是NP难度 (NP hard) 的证明策略:

(1) 选择一个已经证明是NP难度问题 $Q_1$ ;

(2) 求证 $Q_1 \propto Q$ 。

则问题Q是NP难度的。

- 由于 $Q_1$ 是NP难度的，因此所有NP类问题都可约化到 $Q_1$ 。
- 根据约化的传递性，任何NP类问题都可约化到Q。
- 所以，Q是NP难度的。

在此基础上，若进一步表明Q本身是NP类的，则问题Q是NP完全的。

# 10.2 Cook定理和证明

## 10.2.1 Cook定理

斯蒂芬·库克 (Steven Cook) 于1971年证明了第一个NP完全问题，称为Cook定理，表明可满足性问题是NP完全的。

至今至少已有300多个问题被证明是NP难度问题，但尚未证明其中任何一个是属于P的。



## 10.3 一些典型的NP完全问题

证明(一个猜想可能是NP难度的)问题Q确实是NP难度(或NP完全)问题的具体步骤:

——利用多项式约化(归约)的方法

- (1) 选择一个已知其具有NP难度的问题 $Q_1$ ;
- (2) 证明能够从 $Q_1$ 的一个实例 $I_1$ , 在多项式时间内构造Q的一个实例I;
- (3) 证明能够在多项式时间内从I的解确定 $I_1$ 的解。
- (4) 从(2)和(3)可知,  $Q_1 \propto Q$ ;
- (5) 从(1)和(4)及约化的传递性得出所有NP类问题均可约化到Q, 所以Q是NP难度的。
- (6) \*如果Q是NP类问题, 则Q是NP完全的。



## 10.3.1 最大集团

最大集团判定问题是NP类问题。

“集团”是无向图中的完全子图，任意一对顶点间有边相连。

P231 程序10-3是求解该问题的多项式时间不确定算法。



或：  
对给定的图 $G=(V,E)$ ，检查顶点集 $V' \subseteq V$ 中每一对顶点 $u,v$ 间是否存在边 $(u,v) \in E$ ，即可在多项式时间内完成对 $V'$ 是否是集团的检查。

下面证明：**最大集团判定问题是NP完全的。**

证明思路：

◆证明**CNF可满足性** $\propto$ **最大集团判定问题**，所以最大集团判定问题是**NP难度**的。

◆又因为**最大集团判定问题是NP类**问题（前面已证）

所以**最大集团判定问题是NP完全**的。



## 定理10-3 CNF可满足性 $\propto$ 最大集团判定问题

证明：

- 1、在多项式时间内，以任意给定的CNF公式F为输入，构造一个相应的无向图G；
- 2、证明F是可满足的，当且仅当G有一个规模至少为k的集团。



## 定理10-3 CNF可满足性 $\propto$ 最大集团判定问题

证明:

1、在多项式时间内, 以任意给定的**CNF公式F**为输入, **构造**一个相应的**无向图G**;

令 **$F = C_1 \wedge C_2 \wedge \dots \wedge C_k$** 是一个具有**k个子句**的**CNF形式**的布尔公式。

由公式F构造

$V = \{ \langle \sigma, i \rangle \mid \sigma \text{ 是 } C_i \text{ 中的一个文字} \}$

$E = \{ (\langle \sigma, i \rangle, \langle \delta, j \rangle) \mid i \neq j \text{ 且 } \sigma \neq \bar{\delta} \}$

$\sigma$ 和 $\delta$ 处于  
不同的分句中

$\sigma$ 和 $\delta$ 相应的文  
字是一致的

属于哪个子句

边集中为何没有边  
 $(\langle x_1, 1 \rangle, \langle \bar{x}_2, 1 \rangle)$ ?

$G=(V,E)$ 的方法为:

的一个文字},

$E=\{(\langle \sigma, i \rangle, \langle \delta, j \rangle) | i \neq j \text{ 且 } \sigma \neq \bar{\delta}\}$ 。

$F \wedge C_2 =$

边集中为何没有边  
 $(\langle x_1, 1 \rangle, \langle \bar{x}_1, 2 \rangle)$ ?

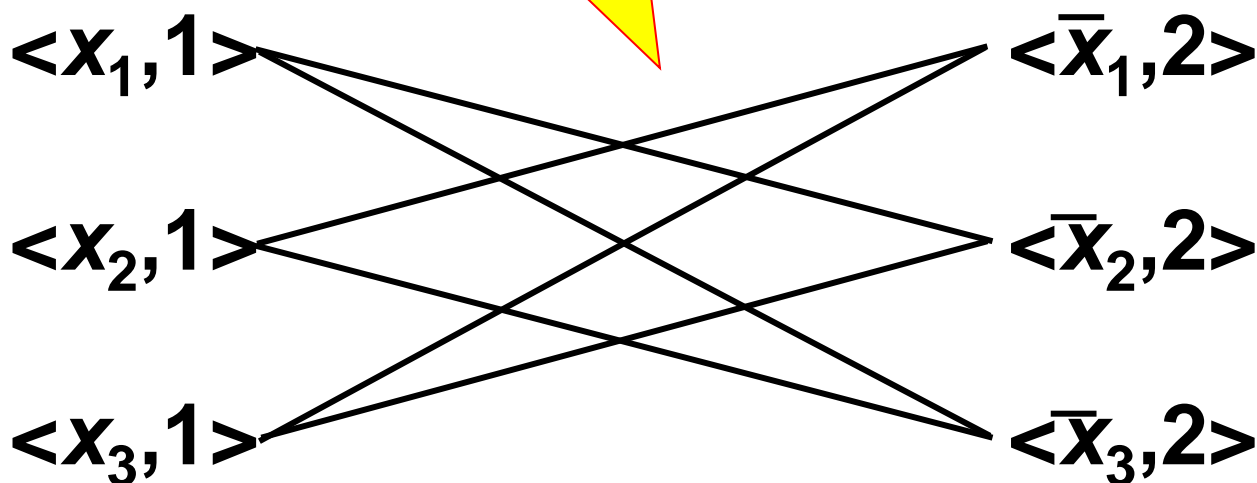
$V = \{ \langle x_1, 1 \rangle, \langle x_2, 1 \rangle, \langle x_3, 1 \rangle, \langle \bar{x}_1, 2 \rangle, \langle \bar{x}_2, 2 \rangle, \langle \bar{x}_3, 2 \rangle \}$

$E = \{ (\langle x_1, 1 \rangle, \langle \bar{x}_2, 2 \rangle), (\langle x_1, 1 \rangle, \langle \bar{x}_3, 2 \rangle),$   
 $(\langle x_2, 1 \rangle, \langle \bar{x}_1, 2 \rangle), (\langle x_2, 1 \rangle, \langle \bar{x}_3, 2 \rangle),$   
 $(\langle x_3, 1 \rangle, \langle \bar{x}_1, 2 \rangle), (\langle x_3, 1 \rangle, \langle \bar{x}_2, 2 \rangle) \}$

$$V = \{ \langle x_1, 1 \rangle, \langle x_2, 1 \rangle, \langle x_3, 1 \rangle, \langle \bar{x}_1, 2 \rangle, \langle \bar{x}_2, 2 \rangle, \langle \bar{x}_3, 2 \rangle \}$$

$$E = \{ (\langle x_1, 1 \rangle, \langle \bar{x}_1, 2 \rangle), (\langle x_1, 1 \rangle, \langle \bar{x}_2, 2 \rangle), (\langle x_1, 1 \rangle, \langle \bar{x}_3, 2 \rangle), (\langle x_2, 1 \rangle, \langle \bar{x}_1, 2 \rangle), (\langle x_2, 1 \rangle, \langle \bar{x}_2, 2 \rangle), (\langle x_2, 1 \rangle, \langle \bar{x}_3, 2 \rangle), (\langle x_3, 1 \rangle, \langle \bar{x}_1, 2 \rangle), (\langle x_3, 1 \rangle, \langle \bar{x}_2, 2 \rangle), (\langle x_3, 1 \rangle, \langle \bar{x}_3, 2 \rangle) \}$$

这种构造能够在多项式时间内完成。



集团判定问题实例

### 定理10-3 CNF可满足性 $\propto$ 最大集团判定问题

证明：

2、证明F是可满足的，当且仅当G有一个规模至少为k的集团。

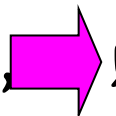
(一方面，如果F可满足  $\Rightarrow$  那么图G中必定存在规模为k的集团。)

若F是可满足的，则必定存在布尔变量的一个赋值，使F的每个子句 $C_i$ 中至少有一个文字为真。若 $\sigma_i$ 是子句 $C_i$ 中为真的文字，则 $S = \{ \langle \sigma_1, 1 \rangle, \langle \sigma_2, 2 \rangle, \dots, \langle \sigma_k, k \rangle \}$ 是图G中相应顶点集合。根据图的构造方法，集合S中任意一对顶点 $\langle \sigma_i, i \rangle$ 和 $\langle \sigma_j, j \rangle$ ，由于 $\sigma_i$ 和 $\sigma_j$ 都为真 ( $\sigma_i \neq \bar{\sigma}_j$ ) 且 $i \neq j$ ，因此它们之间应该有边相连，从而形成完全图。S就是图G的规模为k的集团。

### 定理10-3 CNF可满足性 $\propto$ 最大集团判定问题

证明：

2、证明F是可满足的，当且仅当G有一个规模至少为k的集团。

(另一方面，若图G有一个规模至少为k的集团，则必定存在一种布尔变量赋值，使命题公式F为真，即F是可满足的。)

若图G中存在一个规模至少为k的集团， $S=\{<\sigma_1,1>,<\sigma_2,2>,...,<\sigma_k,k>\}$ 是集团的顶点集合，则必有 $\sigma_i$ 和 $\sigma_j$ 值相同且 $i \neq j$ （否则顶点 $<\sigma_i,i>$ 和 $<\sigma_j,j>$ 之间没有边）。

于是，对S中所有的文字赋真值，对不属于S的变量取任意值，则使得F的每个子句 $C_i$ 中至少有一个文字为真，从而F为真。



虽然当前只是将CNF公式规约成了带某种特定结构的集团实例，仅证明了在这种受限的情况下，最大集团判定问题是NP难度（NP完全）的。但是，这一证明足以证明在一般的图中，最大集团判定问题也是NP难度的。

因为：如果我们有一个多项式时间的算法，它能在一般的图上解决最大集团判定问题，那么它就能在受限的图上解决这一问题。

# 作业

P226      10-6   10-7

10-6 什么是P类和NP类问题？什么是多项式约化？

10-7 什么是NP难度和NP完全问题。



## 附：程序10-3 最大集团判定问题的不确定算法

```
void Clique(int g[][mSize],int n,int k)
{ S=∅;
  for (int i=0;i<k;i++)    //选择k个顶点
  {
    int t=Choice(0,n-1); //任意选择一个顶点t
    if (t∈S) Failure();  //若顶点t已经在S中，则失败终止
    S=S∪{t};
  }
  for (对所有(i,j),i∈S,j∈S且i≠j)    //验证此k个顶点是否
    if ((i,j) ∉ E) Failure();        //形成完全子图
  Success();
}
```