组合最优化

中国科学院大学

数学科学学院

郭田德

电话：88256412

Email: tdguo@ucas.ac.cn

第三章 算法与计算复杂性

3.1 两个问题
3.2 计算复杂性的概念
3.3 算法及复杂性
3.4 多项式时间算法与指数时间算法
3.5 本课程讨论的两类问题

3.1 Two Problems

一个石油公司有一个油田，包括47个钻井平台，这些钻井平台远离Nigeria 海岸。每一个平台都有一组控制，使得与平台相连的油井能以一定的速度将原油输送到海岸的油箱中。为了使得油井能以正常的流速将原油输送出来，需要周期性地去访问一些确定的平台。这项工作需要直升飞机去完成。直升飞机从海岸上的基地起飞，飞向那些要去访问的平台，最后再飞回岸上的基地。

直升飞机的飞行费用很高。石油公司希望设计一个飞行路线，使得直升飞机飞过这些平台，最后回到基地，且飞行时间最短。如果我们假设飞行时间与飞行距离成正比，那么这个问题就变成了一个“欧氏距离旅行商问题”(Euclidean traveling salesman problem)的实例。

在欧氏平面上给定一个点集合V，每一个点有一个坐标(x, y)，两点(x_1, y_1)和(x_2, y_2)之间的距离为$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$。我们的目的是找一个circuit 或 tour，经过集合V中的所有点，且所经过的总长度最小。我们称这个 tour 是一个最优解。

在上面的情况中，点集合V包含了要访问的平台，再加上岸上的基地。

- The Traveling Salesman Problem
- *Euclidean Traveling Salesman Problem*
- *Input*: A set *V* of points in the Euclidean plane.
- *Objective*: Find a simple circuit passing through the points for which the sum of the lengths of the edges in minimized.

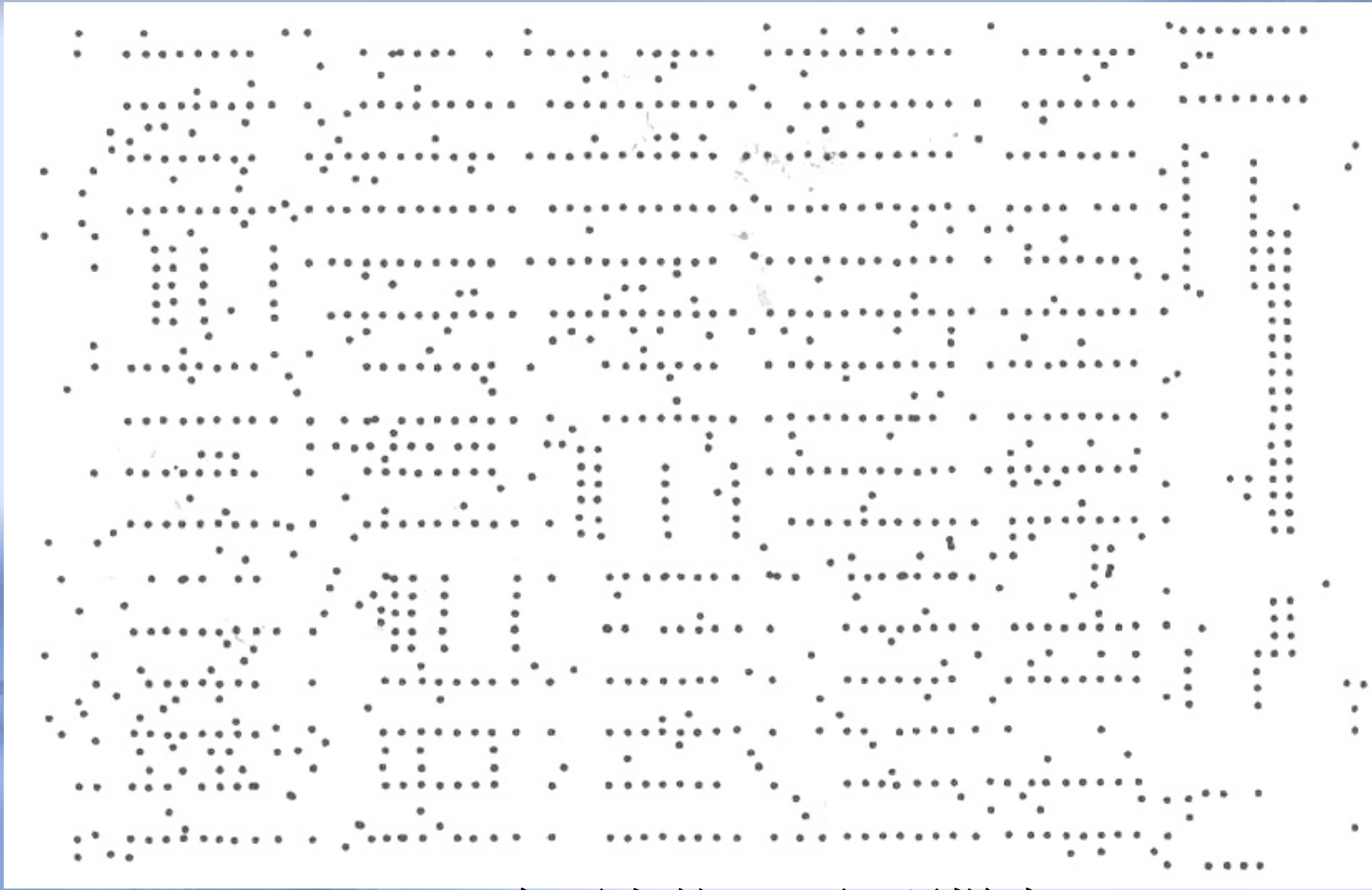
- Methods to solve this problem:

  (1) Try all possible solution: $(n-1)!/2$ different possible solutions.

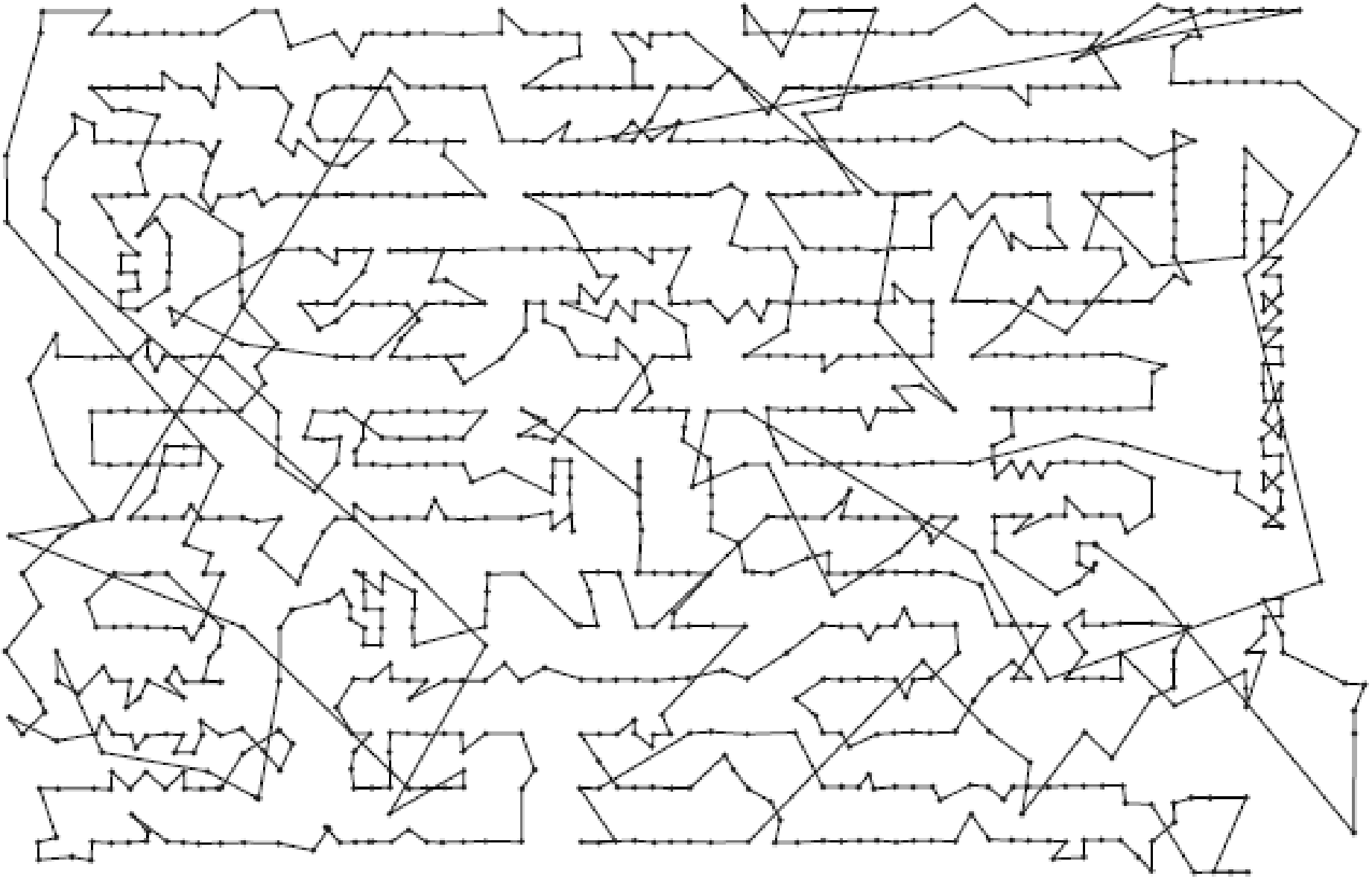
  (2) The Nearest Neighbor Algorithm:

  (i) it is easy to omit a point, which must be visited later at great expense.

  (ii) at times you may “paint yourself into a corner” where you are forced to make a long move to reach the nearest point where you can continue the tour.



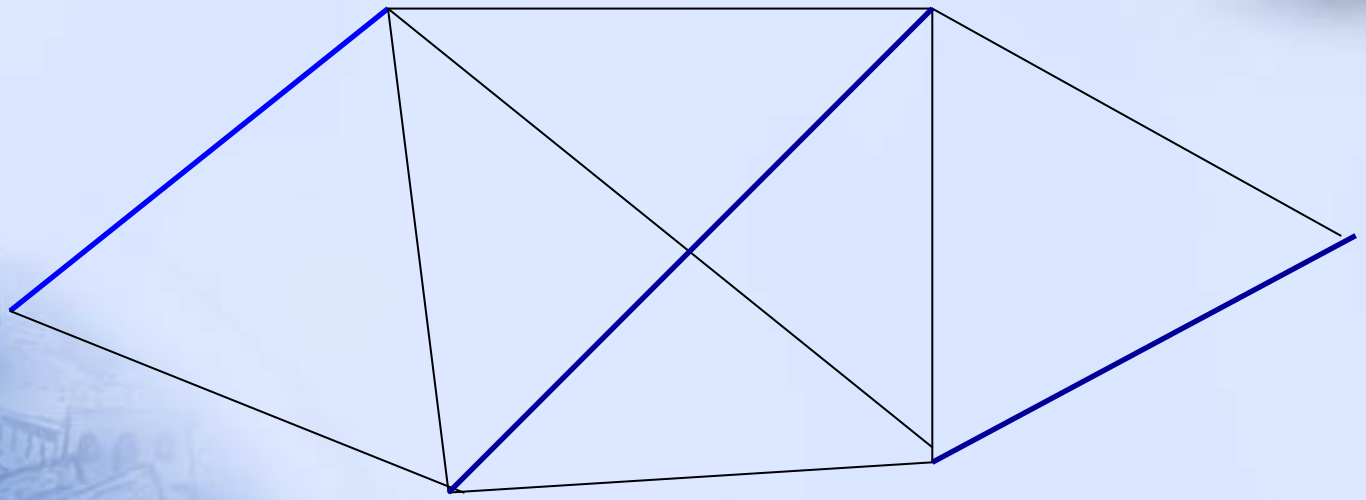
1173个顶点的TSP问题样本



1173个顶点的TSP样本最近邻算法的计算结果

- ✓ The Matching Problem
- ✓ *Euclidean Matching Problem*
- ✓ *Input*: A set *V* of points in the Euclidean plane.
- ✓ *Objective*: Find a set of lines, such that each point is an end of exactly one line, and such that the sum of the lengths of the lines is minimized.

例如：



Some Similarities and Differences:

The *Euclidean Traveling Salesman Problem* and *Euclidean Matching Problem* are two prominent models in combinatorial optimization. The two problems have several similarities.

Similarities:

1. Selecting sets of lines connecting points in the plane;
2. The number of feasible solutions is far too large to consider them all in a reasonable amount of time;
3. Most simple heuristics for the problems do not perform very well.

There is a major difference between the problems lurking under the surface, however.

Differences:

1. There exists an efficient algorithm to find an optimal solution for any instance of the *Euclidean Matching Problem*;
2. Not only is no such algorithm known for the *Euclidean Traveling Salesman Problem*, but most researchers believe that there simply does not exist such an algorithm.

- 本门课程，我们将重点放在这两类问题之间的分界线上：已知有有效算法的问题和象**Euclidean Traveling Salesman Problem**一样难的问题
- 多数的努力都花费在描述分解线“好”的一边的问题模型上，除了它们本身的重要性外，这些“好”的模型还给我们提供了建筑模块，来解决“坏”的一边的问题的途径。

3.2 计算复杂性的概念

- 对于那些不确定的非数学型任务，根本不可能用计算机解决。
- 对有些问题，虽然原则上存在一种算法可求解其任一实例，但因该算法需要过长的时间或太大的存贮空间而使它变得完全无用。
- 对于绝大多数问题：求解某一问题的不同算法在时间、空间要求上相差很大，即使同一算法，当其用来求解问题的不同实例时，其性能表现差异也较大。
- 不同的问题往往要设计不同的算法，且同时又想用同一种算法去尽可能多地求解不同类型的问题。

如何解释这些错综复杂、多种多样的现象呢？

能否给出一个一般的划分与衡量标准，以区分不同问题的难易程度、度量不同算法有效性之间的差异？

- 算法复杂性理论试图从一般角度去分析实际中各种不同类型的问题，通过考察可能存在的求解某个问题不同算法的复杂性程度来衡量该问题的难易程度，由此将问题划分为不同的类型，并对各种算法按其有效性进行分类。
- 达到这一目的的主要方法就是分析求解问题算法的计算复杂性。
- 计算复杂性回答的是求解问题所需要的各种资源的量，它主要考虑的是设计可以用于估计、定界任一算法求解某些类型的问题时所需的和仅需的计算资源量的技术或方法。

3.3 算法及复杂性

- 所谓一个问题（**problem**）是指一个有待回答的，通常含有几个其值还未确定的自由变量的一个一般性提问**(question);**
- 它由两部分决定：一是对其所有参数的一般性描述；二是对该问题的答案所应满足的某些特性的说明；
- 一个问题的某个例子则可通过指定问题中所有参数的具体取值来得到。在以下论述中，我们将经常用符号Π来表示某个问题，而其例子将用**I**表示。

典型的旅行商问题

该问题的参数是由所需访问城市的一个有限集合 $C = \{C_1, C_2, \ldots, C_m\}$ 和对 C 中每一对城市 C_i, C_j 之间的距离 $d(C_i, C_j)$ 所组成。它的一个解是对所给城市的一个排序 $< C_{\pi(1)}, C_{\pi(2)}, \cdots, C_{\pi(m)} >$，这里 $(\pi(1), \pi(2), \cdots, \pi(m))$ 表示 {1,2, …， m} 的某种排序，使得该排序极小化下列表达式的值

$$\sum_{i=1}^{m-1} d(C_{\pi(i)}, C_{\pi(i+1)}) + d(C_{\pi(m)}, C_{\pi(1)}).$$

这个表达式定义了由 $C_{\pi(1)}$) 开始，依次访问排列中的每个城市，并最后从城市 $C_{\pi(m)}$ 直接返回到 $C_{\pi(1)}$) 所构成的环游的长度。旅行商问题的任一个例子是通过限定城市的数目，并指定每两个城市之间的具体距离而得到的。

对$d(C_i, C_j)$的约束是：

- （1）$d(C_i, C_j)$非负，即$d(C_i, C_j) \geqslant 0, 1 \leqslant i,j \leqslant n$;
- （2）对角线上的元素为0，即$d(C_i, C_i) \geqslant 0$, $1 \leqslant i \leqslant n$;
- （3）对称性，即$d(C_i, C_j) = d(C_j, C_i)$, $1 \leqslant i,j \leqslant n$;
- （4）三角不等式，即$d(C_i, C_j)+d(C_j, C_k) \geqslant d(C_i, C_k)$, $1 \leqslant i,j,k \leqslant n$;

所有可行解的集合构成解空间S，即

$S=\{n$个城市的所有循环排列$\}$。

解空间的规模为$|S|=\dfrac{(n-1)!}{2}$。

旅行商问题的任一个例子是通过限定城市的数目，并指定每两个城市之间的具体距离而得到的。

0－1 背包问题

一个旅行者从 n 种物品种选取 b 公斤重的物品，每种物品至多选一件。问这个旅行者应该如何选取，使得所选物品的总价值最大。

设第 i 种物品的重量为 w_i，价值为 c_i，$i=1,2,\cdots,n$，则问题是

$$\max z=\sum_{i=1}^{n} c_i x_i$$

$$s.t. \quad \sum_{i=1}^{n} w_i x_i \le b_i$$

$$x_i = 0,1$$

最大截问题

设 $G=(V,E)$ 是一个无向图，把节点集合 V 分划为两个子集 V_0,V_1，使得图中那些节点分别在 V_0 和 V_1 中的边集合的权和最大。

设 $\delta(V_0,V_1)$ 是一个划分，$\varpi(u,v)$ 表示边 (u,v) 的权，则问题模型为：

$$\max f(V_0,V_1)=\sum_{(u,v)\in\delta(V_0,V_1)}\varpi(u,v)$$

$$\text{s.t.}\quad \delta(V_0,V_1)=\{(u,v)\in E:u\in V_0,v\in V_1\}$$

图的着色问题

设$G=(V,E)$是一个无环图，对G的每个节点着色，使得任意两个相邻的节点都有不同的颜色，要求所用的颜色数最少。

图$G=(V,E)$中同一种颜色的节点集合是$G=(V,E)$的一个独立集。若$G=(V,E)$的节点集合V可划分为k个独立集$V_1,V_2,\cdots,V_k$，则$G=(V,E)$是k可着色的。因此，图的着色问题等价于找一个映射$f:V\to\{1,2,\cdots,k\}$。设Δ是$G=(V,E)$的最大度数，$\chi(G)$是$G=(V,E)$的色数（使G最优着色的颜色数），则问题是

$$\min k,\ \chi(G)\le k\le\Delta+1$$

$$s.t.\quad \forall u,v\in V:(u,v)\in E\Rightarrow f(u)\neq f(v)$$

- 所谓算法（**algorithm**）是指可用来求解某一问题的、带有一般性的一步一步的过程。
- 算法是用来描述可在许多种计算机上实现的任一计算流程的抽象形式，其一般性可以超越任何具体实现时的细节。
- 称一个算法求解一个问题Π，是指该算法可应用到Π的任一例子**I**，并保证总能找到该例子的一个解。
- 对于同一问题，常常有几个不同的算法可以求解它，问：什么是求解问题的一个“好”算法？
- 要回答这一问题，我们就需要有一个比较不同算法好坏或相对有效性的方法。

- 一个算法的有效性可以用执行该算法时所需要的各种计算资源的量来度量。
- 最典型、也是最主要的两个资源就是所需的**运行时间和内存空间**。
- 通常总是将最快的算法与最有效的算法等同起来。
- 在复杂性研究中，衡量一个算法的效果，最广泛采用的标准是在产生最终答案前它所花费的时间，并常常**称复杂性为时间复杂性**。
- 同一算法所需时间多少随着计算机的不同可能有很大的差别。
- 同一算法和同一计算机，当用它来求解某一问题的不同例子时，由于有关参数取值的变化，使得所需运行时间也有较大差别

- 计算模型：如各种形式的图灵（Turing）机，随机存取机(random-access machines，常简写为RAMs)等，然后基于这些计算模型来研究算法。
- 假设每做一次初等运算均需要一个单位时间
- 用算法在执行过程中总共所需要的初等运算步数来表示算法用于求解任一问题的某一例子时所需要的时间。
- 所谓的初等运算是指：算术运算、比较和转移等最基本的操作。

- 问题例子大小（size）：所谓一个问题例子的大小是指为描述或表示它而需要的信息量。
- 例子大小的值与其求解的难易程度成正比，并称相应的表示方法为编码策略(encoding scheme)。
- 作为输入所提供给计算机的任一问题例子的描述可以看作是从某一有限字符表中选取所需字符而构成的有限长字符串。称该有限字符表中的字符为编码，而由其中之字符如何组成描述问题例子的字符串的方法则称为编码策略。

- 合理编码策略应满足两个基本要求，即可解码性（decodability）与简洁性（conciseness）
- 可解码性是指对问题例子的任一特定组成部分或分量，我们应能够指定一有效的算法，它可从任何给定的已经被编码的例子中来提取出对那个分量的一个描述。
- 而编码策略的简洁性意味着它满足下列两个条件：(a) 一个例子的编码应是简洁的，而不应被某些不必要的信息或符号充斥或拉长。(b) 例子中所出现的数字应统一用二进制、十进制或任何大于1的数为固定基来表示。
- 一个典型的方法就是利用所谓的结构化字符串，通过递归、复合的方式来给出所考虑问题的合理编码策略。

问题的输入长度：给定任一问题Π，假设已找到描述该问题例子的一个合理编码策略，则对Π的任一例子I，称其依编码策略e所得的相应字符串描述中所含字符的个数为其输入长度，并将该输入长度的具体数值作为例子I的大小的正式度量。

3.4 多项式时间算法与指数时间算法

- 时间复杂性函数（time complexity function）的定义：

  对某一问题 Π 和任一可能的输入长度，如为 n，称用所给算法求解 Π 的所有大小为 n 的例子所需的时间的最大值为该算法在输入长度为 n 时的复杂性。

- 在计算机科学中存在着这样一个一般的约定：仅当算法的时间复杂性函数随着问题例子输入长度的增加而多项式地增长时，才认为这个算法是实用的、有效的。按照这种观点，则可以将算法分为两大类。

在给出两类算法的定义之前，先引进在表示算法复杂性时常用的、类似于数学中同阶大小的记号O(·)，对于定义于正整数集上的两个正实值函数$f(n)$与$g(n)$，若存在两个常数：$c > c' > 0$,使得当n充分大时有$c'g(n) \le f(n) \le cg(n)$，则记$f(n) = \mathrm{O}(g(n))$，利用O(·)可将函数划分为不同的类，在复杂性理论中，对如此定义的同一类型的不同函数往往不加区分。

多项式时间算法（polynomial time algorithm）：是指存在某个以输入长度 n 为变量的多项式函数 $p(n)$，使其时间复杂性函数为 $O(p(n))$ 的算法。因此，复杂性为 $O(n)$,$O(10^6n^3)$,$O(5n^8)$等的算法均为多项式时间算法。

指数时间算法（exponential time algorithm）：是指任何其时间复杂性函数不可能如上用多项式函数去界定的算法。这类算法的时问复杂性函数典型的例子有 $2^n, n!, n^n, 2^{n^2}, n^{\log n}, n^{n^n}$ 等。严格地说，n^{n^n} 并不是我们通常说的指数函数，它的增长速度比指数还快。在复杂性理论中，我们将所有这些类型的函数统统称为指数函数而不做进一步细分。

- 算法的复杂性对计算机的解题能力（速度和规模）有重大影响。
- TSP问题：枚举法可能的路径数为(n-1)!/2，若以路径间的比较运算为基本操作，则需要的基本操作数为：(n-1)!/2-1。用每秒1亿次浮点运算的计算机进行求解，n=10，t=0.0018s，而n=20，t=19.29年！
- 0-1背包问题：枚举法需要比较的解数为2^n-1，n=10，t=0.01s，而n=60，t=366世纪！

时间复杂性函数	规　模 n					
	10	20	30	40	50	60
n	0.00001 秒	0.00002 秒	0.00003 秒	0.00004 秒	0.00005 秒	0.00006 秒
n^2	0.0001 秒	0.0004 秒	0.0009 秒	0.0016 秒	0.0025 秒	0.0036 秒
n^3	0.001 秒	0.008 秒	0.027 秒	0.064 秒	0.125 秒	0.216 秒
n^5	0.1 秒	3.2 秒	24.3 秒	1.7 分	5.2 分	13.0 分
2^n	0.001 秒	1.0 秒	17.9 分	12.7 天	35.7 年	366 世纪
3^n	0.059 秒	58 分	6.5 年	3855 世纪	$2*10^8$ 世纪	$1.3*10^{23}$ 世纪

1天内可解决的最大问题实例的规模

时间复杂性函数	用现在的计算机	用快 100 倍的计算机	用快 1000 倍的计算机
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
n^5	N_4	$2.5N_4$	$3.98N_4$
2^n	N_5	$N_5+6.64$	$N_5+9.97$
3^n	N_6	$N_6+4.19$	$N_6+6.29$

常称一个问题具有难解性（**intractability**）或为难解的，如果它是如此困难，以至于没有多项式时间算法可以去求解它。由定义，一个问题的难解性可以由下列原因之一引起：

- 对于某些意义明确的数学问题，它是如此困难，使得根本就不存在算法，即其不可能用任何算法，更不用说多项式时间算法来求解。故其在更强意义下为难解的，称这类问题为不可判定型（undecidability）问题。典型的不可判定问题包括著名的停机问题（给定任一计算机程序，及其输入，它会停止吗？）和Hilbert第十问题（整系数多项式方程组的可解性）。
- 据此，可以将所有的问题分为两大类：不可判定类型的和可判定类型的。对于可判定型问题，在原则上总存在一个算法，可以解决该问题的任何一个实例。在计算复杂性理论中，主要是研究求解可判定型问题算法的复杂性。

- 问题的解本身是如此庞大以致于不可能用问题例子输入长度的一个多项式函数来界定描述解的表达式之长度。例如，考虑旅行商问题的变形，当我们的目的是要求给出所有总长不超过某一阈值的所有旅行路线，而该阈值大于问题最优旅行路线的长度时，就遇到这一情况。通常，这种情况的存在性可由问题的定义容易看出，且常常意味着原问题没有被现实合理地定义。
- 最后一个原因，也就是我们通常所想到的，即问题太困难，要找到它的一个解就需要用指数时间的算法。

多项式时间算法与指数时间算法的以下几点说明：

- 所定义的时间复杂性为最坏情形度量。
- 在考虑算法的复杂性时，通常只关心算法在问题例子的规模*n*充分大时的表现。故对小规模问题，有些指数时间算法可能要比多项式时间算法好。
- 关于问题的难解性、一个算法是多项式时间算法还是指数时间算法等，本质上并不依赖于特定的编码策略和具体的计算模型。

3.5 本课程讨论的两类问题

- 本课程主要讨论的第一类问题：有多项式算法的问题。
- 讨论的第二类问题是：是否存在多项式算法还是一个**open**问题。
- 但是，我们知道，如果第二类问题中的一个有一个多项式算法，那么这类问题中的所有问题都有多项式算法。
- **Eulidean Matching Problem**属于第一类；
- **Eulidean TSP**问题则属于第二类。

- 对于第一类问题，我们的目标当然是寻找尽可能快的多项式算法；
- 对于第二类问题，尽管人们相信没有多项式算法，但常常有比枚举算法好的多的算法。另外，对许多问题，人们可以找到近似算法是多项式的，并且保证所得到的解与精确解在一定的比例范围内。
- 许多第二类问题具有重大的实际意义，需要找出兼顾解得质量以及运行时间的较好算法。一种方法是设计平均性能良好的概率算法，这种算法在多项式时间里几乎总是产生最优解，但在最好情况下，仍然需要指数界的时间。另一种方法是设计求近似最优解的近似算法，这种方法采取的策略和启发式简单、直接，而且对某些问题能够产生很好的结果。