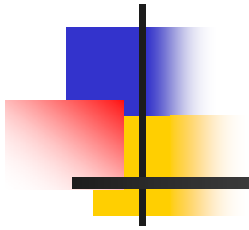


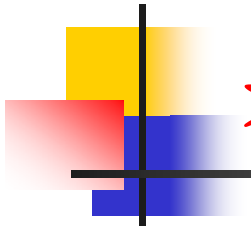
智能优化方法
Nature Inspired Computation

李 斌

binli@ustc.edu.cn

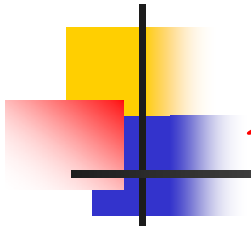
智能信息处理实验室

自然计算与应用实验室



考核方式

- 课程设计报告
  - 用一种（或多种）智能优化方法，实现实际或者虚拟优化问题的求解。
  - 鼓励与本人未来研究领域的结合。
  - 报告形式：小论文或实验报告的形式，要求包含：实验目的、技术方案、量化的实验结果、对结果的简单分析（或解释）。
  - 元旦之前，将1份打印件放到我的信箱里。



第一章 概述

- 最优化问题的定义
- 最优化问题的分类
- 关于计算复杂性
- 最优化方法的一般结构

最优化问题的定义

最优化问题的一般形式为

$$\min f(x)$$

$$s.t.\ x \in X$$

其中$x \in R^n$是决策变量，$f(x)$为目标函数，$X \subset R^n$为约束集或可行集。特别地，如果约束集 $X = R^n$，则最优化问题称为无约束最优化问题

$$\min_{x \in R^n} f(x)$$

最优化问题的定义

约束最优化问题通常写为

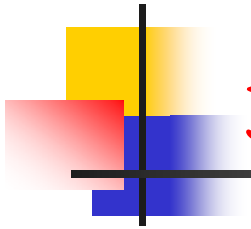
$$\begin{aligned} &\min f(x) \\ &s.t.\ c_i(x) = 0, \quad i \in E, \\ &\quad\ c_i(x) \geq 0, \quad i \in I. \end{aligned}$$

这里E和I分别是等式约束的指标集和不等式约束的指标集，$c_i(x)$是约束函数.

最优化问题的分类

按照运筹学的观点分类：

- 线性规划
- 非线性规划
- 整数规划
- 动态规划
- 多目标规划
- 。。。。。。



最优化问题的分类

从应用的角度分类:

- 函数优化
- 组合优化
- 可靠性设计问题
- 调度问题
- 高级运输问题
- 网络设计与路径
- ……

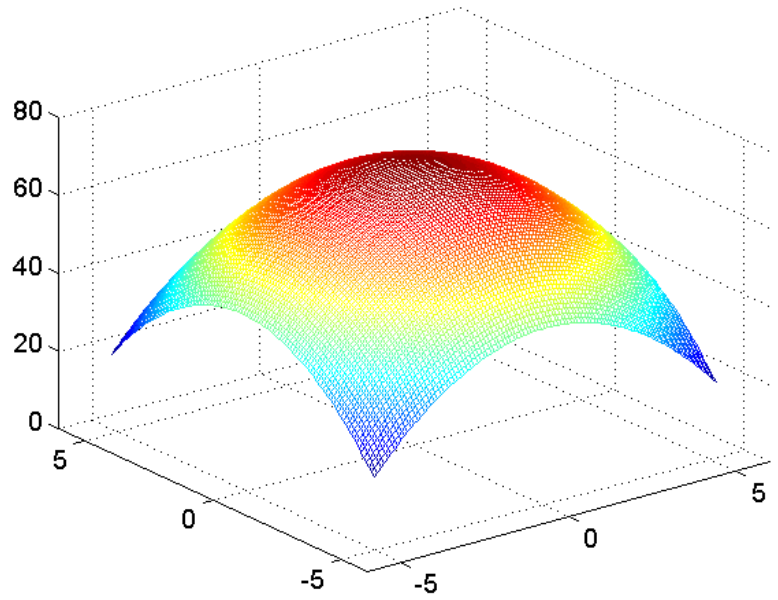
最优化问题举例(1)

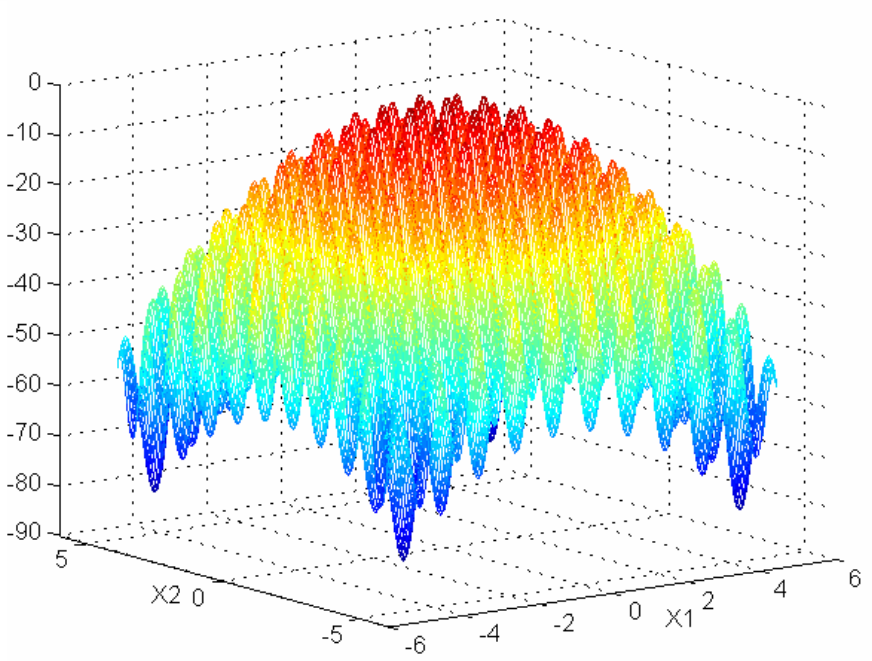
- 函数优化

令 S 为 R^n 上的有界子集，$f: S \rightarrow R$ 为 n 维实值函数，所谓函数 f 在 S 域上全局最大化就是寻求点 $X_{max} \in S$ 使得

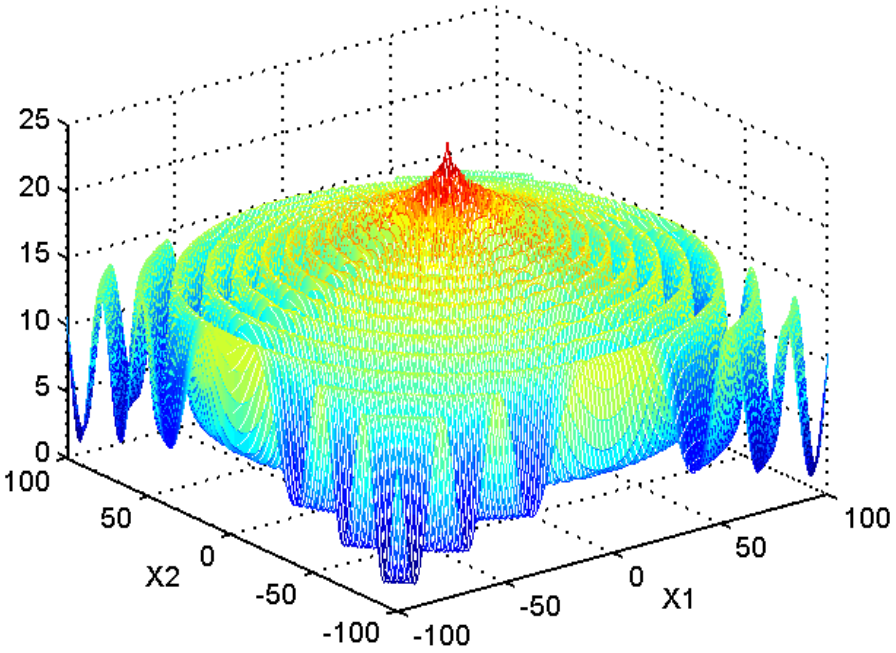
$$\forall x \in S : f(X_{\max}) \geq f(x)$$

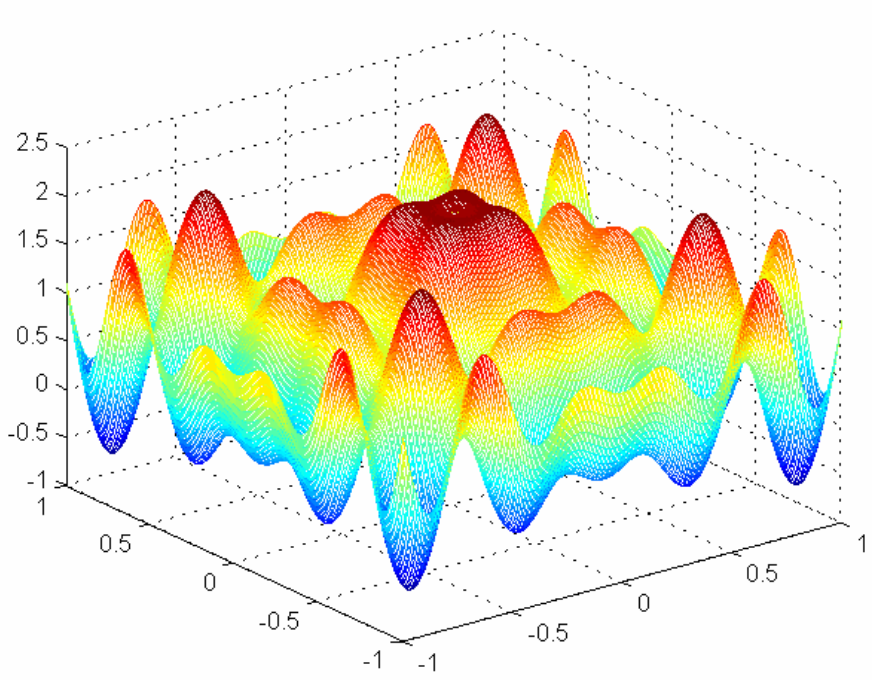
函数优化问题



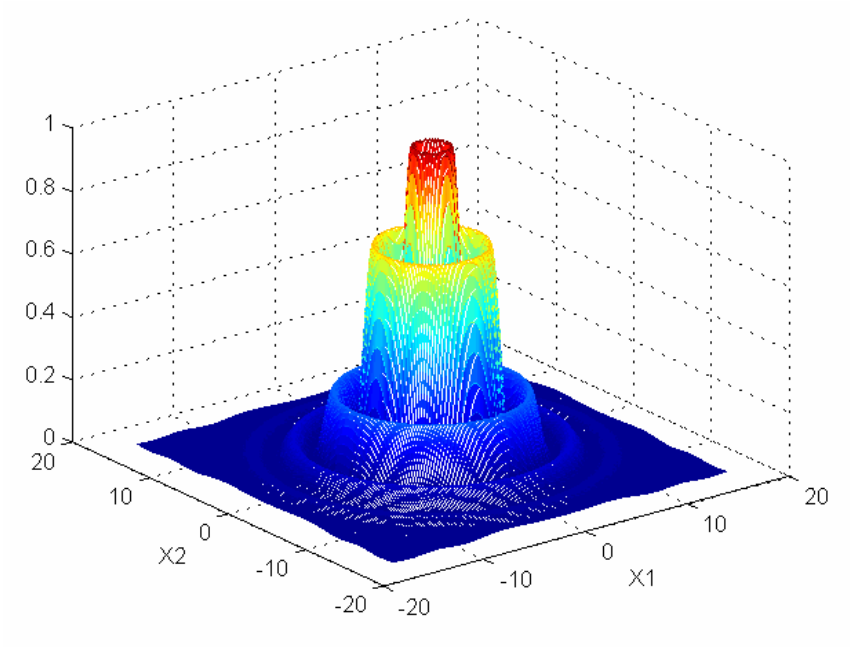


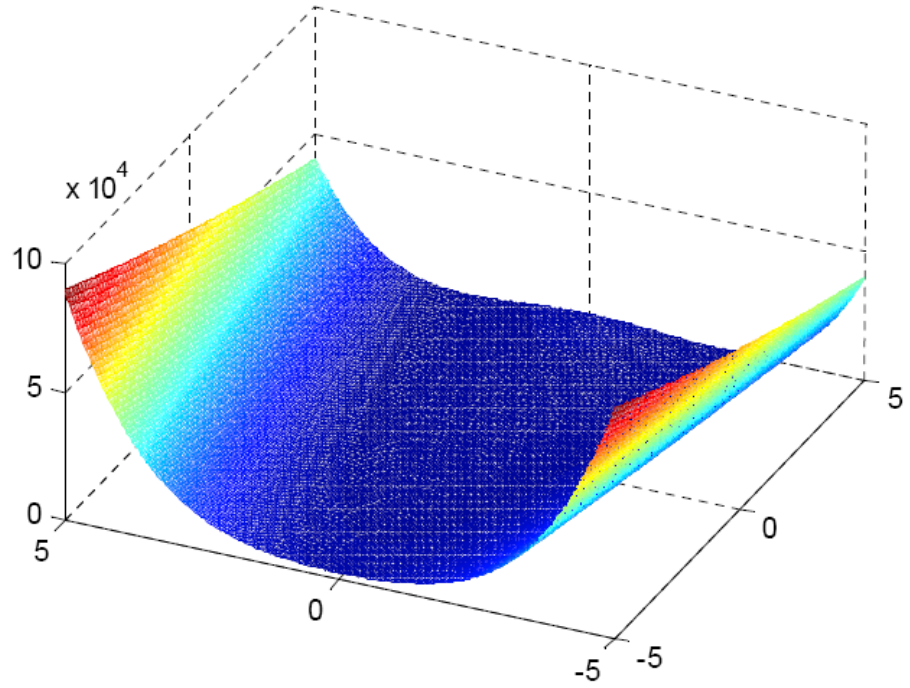
函数优化问题


25
20
15
10
5
0
100
50
0
-50
-100
X2
-100
-50
0
50
100
X1


2.5
2
1.5
1
0.5
0
-0.5
-1
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

函数优化问题





最优化问题举例(2)

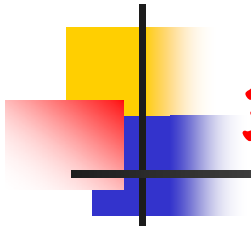
- 组合优化

定义：组合优化问题π是一个极小化问题，或是一个极大化问题，它由下面三部分组成：

(1)实例集合；

(2)对每一个实例 I，有一个有穷的可行解集合 $S(I)$；

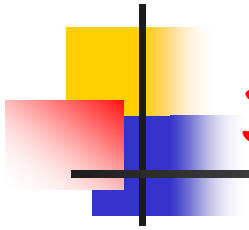
(3)目标函数 f，它对每一个实例 I 和每一个可行解 $\sigma \in S(I)$，赋以一个有理数 $f(I,\sigma)$。



组合优化问题

- 一个通俗的定义:

  所谓组合优化，是指在离散的、有限的数学结构上，寻找一个（或一组）满足给定约束条件并使其目标函数值达到最大或最小的解。一般来说，组合优化问题通常带有大量的局部极值点，往往是不可微的、不连续的、多维的、有约束条件的、高度非线性的NP完全（难）问题，因此，精确地求解组合优化问题的全局最优解的“有效”算法一般是不存在的。



组合优化问题

- 集覆盖问题（set-covering problem）
- 装箱问题（bin-packing problem）
- 背包问题（knapsack problem）
- 指派问题（assignment problem）
- 旅行商问题（traveling salesman problem）
- 影片递送问题（film delivery problem）
- 最小生成树问题（minimum span tree problem）
- 图划分问题（graph partitioning problem）
- 作业调度问题（job-shop scheduling problem）

组合优化问题——集覆盖问题

- 集覆盖问题（set-covering problem）
- 对于一个m行n列的0－1矩阵A，用最小的代价选择一些矩阵的列，使其能够覆盖所有的行。
- 设向量x的元素 $x_j=1$ 表示列 j 被选中（费用是 $c_j>0$），$x_j=0$ 则表示其未被选中（$j=1,2,\ldots,n$）。
- 已经证明集覆盖问题是NP完全问题。

组合优化问题——集覆盖问题

$$\min \quad z(x) = \sum_{j=1}^{n} c_j x_j$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1,2,\cdots,m$$

$$x_j \in \{0,1\}, \qquad j = 1,2,\cdots,n$$

- 如果所有费用c_j都相同，则问题称为单一费用问题（**unicost set-covering problem**）。如果为等式约束，则称为集划分问题（**set partitioning problem**）

组合优化问题——集覆盖问题

$$c = [10 \quad 15 \quad 11 \quad 10 \quad 8 \quad 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$c = [10 \quad 5 \quad 11 \quad 10 \quad 8 \quad 2]$$

$$(a_{ij}) = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$x = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6]$$

$$1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0$$

- **Cost = 29**

表 2.1　集覆盖问题的应用

作　者	应　用
Arabeyre 等(1969)[18]	航线机组成员调度(airline crew scheduling)
Balinski (1965)[39]	开关电路调度(switching circuit scheduling)
Balinski and Quandt (1964)[40]	卡车派遣(trucking dispatching)
Bellmore (1971)[53]	其他例子
Bellmore, Greenberg and Jarvis (1970)[54]	网络攻击与防卫(network attack and defense)
Bellmore (1971)	网络攻击与防卫(network attack and defense)
Busacker and Satty (1965)[82]	图着色(map coloring)
Charnes and Miller (1956)[96]	铁路工作人员调度(railroad crew scheduling)
Cobham, Fridshal and North (1961)[126]	PERT/CPM 分析
	开关电路调度(switching circuit scheduling)
	符号逻辑(symbolic logic)
Day (1965)[150]	信息恢复(information retrieval)
Garey Johnson (1979)[208]	资源分配(resource allocation)
Garfinkel (1968)[209]	政治重新划分选区(political redistricting)
Labordere (1969)[389]	电路设计(circuit design)
Levin (1969)[405]	车辆路径(vehicle routing)
Revelle, Marks and Leibman (1970)[535]	设备定位(facilities location)
Root (1964)[540]	符号逻辑(symbolic logic)

组合优化问题——装箱问题

- 装箱问题（bin packing problem）

用最少的箱子将所有物品都装下

$$\min \quad z(y)=\sum_{i=1}^{n} y_i$$

$$s.t. \quad \sum_{j=1}^{n} w_j x_{ij} \le c y_i, \quad i \in N=\{1,2,\cdots,n\}$$

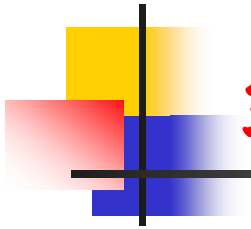
所装物品不得超过箱子得容积

一个物品只能放入一个箱子

$$\sum_{i=1}^{n} x_{ij}=1, \qquad j \in N$$

$$y_i = 0 \text{ 或 } 1, \qquad i \in N$$

$$x_{ij} = 0 \text{ 或 } 1, \qquad i, j \in N$$



组合优化问题——装箱问题

- 货运装箱问题
- 截铜棒问题
- 布匹套裁问题
- 。 。 。
- 装箱问题属于NP－难问题

组合优化问题——背包问题

0/1背包问题： 给出几个体积为S_1，S_2，…，S_n的物体和容量为C的背包；要求找出n个物件的一个子集使其尽可能多地填满容量为C的背包。

数学形式：

最大化 $\sum_{i=1}^{n} S_i X_i$

满足 $\sum_{i=1}^{n} S_i X_i \le C,$

组合优化问题——背包问题

广义背包问题：输入由背包容积C和两个向量：物品体积$S = (S_1, S_2, \ldots, S_n)$和物品价值$P = (P_1, P_2, \ldots, P_n)$组成。设X为一整数集合（物品的标识），$X = 1, 2, 3, \ldots, n$，T为X的子集，则问题就是找出满足约束条件，并使总价值最大的子集T。

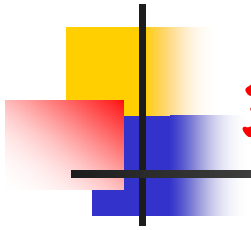
数学形式：

最大化 $\sum_{i=1}^{n} P_i X_i$

满足 $\sum_{i=1}^{n} S_i X_i \le C, \qquad X_i \in \{0,1\}, \quad 1 \le i \le n$

组合优化问题——背包问题

- 在应用问题中，设**S**的元素是***n***项经营活动各自所需的资源消耗，**C**是所能提供的资源总量，**P**的元素是人们从每项经营活动中得到的利润或收益，则背包问题就是在资源有限的条件下，追求总的最大收益的资源有效分配问题。
- 背包问题属于NP-难问题



组合优化问题——背包问题

- 多选择背包问题：有一个容积有限的背包。要放入背包的物品被分为不重叠的若干类，每类中有若干不同的物品，从每类中选择一个物品，使得物品总体积在满足背包容积约束的前提下最大化收益。
- 属于NP－难问题

组合优化问题——背包问题

$$\max \quad \sum_{i=1}^{m}\sum_{j=1}^{n_i} c_{ij}x_{ij}$$

最大化所选物品的总价值

$$s.t. \quad \sum_{i=1}^{m}\sum_{j=1}^{n_i} w_{ij}x_{ij} \le W$$

所选物品的总体积不得超过背包容积

每一类中只能选一个物品

$$\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1,2,\cdots,m$$

$$x_{ij} \in \{0,1\}, \quad \forall i, j$$

- i为类下标；j为类中物品的下标；n_i是第i类中物品的数量；c_{ij}是第i类中第j个物品的收益；m是类的数量；w_{ij}为第i类中第j个项目的体积，W是背包的容积。

组合优化问题——旅行商问题

- 旅行商问题（Traveling Salesman Problem）
- 寻找一条最短的遍历n个城市的路径，或者说搜索整数子集$X = \{1, 2, \ldots, n\}$（X的元素表示对n个城市的编号）的一个排列 $\pi(X) = \{v_1, v_2, \ldots, v_n\}$，使下式取最小值。

$$T_d = \sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_1, v_n)$$

- 式中的$d(v_i, v_{i+1})$表示城市v_i到城市v_{i+1}的距离。
- 对称旅行商问题是NP－难问题

组合优化问题——影片递送问题

- 影片递送问题（Film Delivery Problem）
- 一盘影片拷贝要在n个电影院放映。每个电影院放映的次数已定，记为一个非负的整数$d_i(i=1，2，\ldots，n)$。两个影院间的距离记为w_{ij}，若影院i和j不能直接相连，则令$w_{ij}=+\infty$。
- 问题是要为影片递送员找一个巡回，从主影院1开始，将影片拷贝送到第i家影院$d_i(i=1，2，\ldots，n)$次，最后回到主影院1，并极小化总的路线长度。当所有的$d_i(i=1，2，\ldots，n)$为1时，FDP变为经典的TSP。
- FDP是TSP的新扩展，它可以推广到一大类路径与排序问题中。

组合优化问题——最小生成树问题

- 最小生成树问题（Minimum Span Tree Problem）
- 考虑一个连通的无向图G = (V, E)，其中，V = {(v_1, v_2, …, v_n}是代表终端或通信站的有限的端点集。E = {e_{ij} | e_{ij} = (v_i, v_j), v_i, v_j∈V}是代表终端或站间连线的有限边集。每条边有一个记为W = {w_{ij} | w_{ij} = w(v_i, v_j), w_{ij} > 0, v_i, v_j∈V}的正实数的权重，代表距离、价格等。端点和边有时也称为节点和连线。
- 生成树是连接V中所有端点的来自E的最小边集，因此对于图G至少可找到一棵生成树。最小生成树，记为T^*，是边的权重和最小的生成树。其描述如下：

$$T^* = \min_{T} \sum_{e_{ij} \in E} w_{ij}$$

组合优化问题——图划分问题

- 图的二划分问题：对于一无向图G，设其顶点集合为V，将顶点集合V划分为两个子集V1和V2，$V1 \cap V2 = \Phi$，求使V1和V2两顶点子集之间联结最少的一种划分。
- 图的划分问题在电子线路设计中非常重要。例如，在多层印刷电路板的布局设计中，使层间联线数目最少的器件布局等。由于图的划分问题的计算复杂度极高(3000个节点的二划分问题的搜索空间可达10^{900})，因此，在实用规模上精确求出最优解是不可能的。

组合优化问题——调度问题

- 广义地讲，调度问题考虑的是随时间的变换，如何调度有限的资源在执行任务的同时满足特定约束。
- 资源：人力、金钱、机器、工具、材料、能源、等等。
- 任务：制造系统的机器划分；计算机系统的信息处理。
- 任务的表征：完成时间、预期时间、相对紧急权重、处理时间和资源消耗。

组合优化问题——调度问题

古典作业车间调度问题（**Job-shop Scheduling**）：有m台不同的机器和n个不同的工件，每个工件包含一个由多道工序组成的工序集合，工件的工序顺序是预先给定的。每道工序用它所要求的机器和固定的加工时间来表示。此外对工件和机器有一些约束，例如:

(1) 一个工件不能两次访问同一机器;

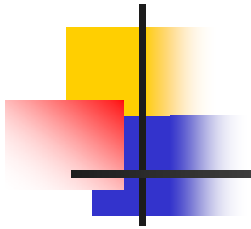
(2) 不同工件的工序之间没有先后约束;

(3) 工序一旦进行不能中断;

(4) 每台机器一次只能加工一个工件;

(5) 下达时间和交货期都不是给定的。

问题：确定机器上工序的顺序，以最小化完成所有工件所需的最小加工持续时间。



- 关于各类优化问题的工程实例，请参考：
- 《遗传算法与工程优化》
  - 玄光男，程润伟著，清华大学出版社

关于计算复杂性

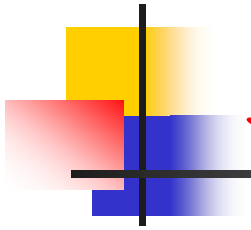

计 算 机 数 学
——计算复杂性理论与NPC、NP难问题的求解
西 安 交 通 大 学 数 学 研 究 生 教 学 丛 书
陈志平 徐宗本 编著
科学出版社

关于计算复杂性

- 利用计算机求解某一类问题的方法，称为计算方法，简称算法。所谓算法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。它是用来描述可在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。
- 可计算理论从建立某个可恰当描述计算机运作原理的计算模型出发，精确定义什么是计算、什么是可计算，再进一步回答某个或某类问题可否用计算机求解的问题。
- 若所给问题可以求解，则称该问题是可计算的或可解的，否则称为不可解。

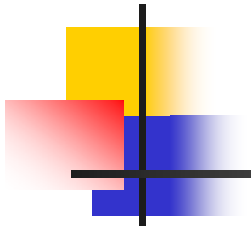
关于计算复杂性

- 计算复杂性理论试图在一般框架下分析各种不同类型的问题，通过考察可能存在的、求解某个问题的、不同算法的复杂性程度，来衡量（求解）该问题的难易程度，如是否很难求解或较易求解等。
- 算法的计算复杂性回答的是求解问题的某一个算法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要的和仅需的计算资源量的技术或方法。
- 对于同一问题，常常有几个不同的算法可以求解它，一个很自然的问题就是，什么是求解该问题的一个“好”算法？



算法的计算复杂性

- 广义地讲，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。
- 计算资源：运行时间，内存空间
- 算法的计算复杂性：时间复杂性，空间复杂性



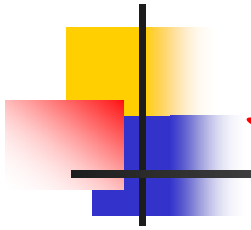
- 如何比较算法的时间复杂性呢?
  - 需要定义一个统一的计算平台模型，以消除不同形式计算机对算法复杂性分析的影响（图灵机、随机存取机。。。）
  - 需要定义一个统一的时间单位（一次初等运算）
  - 相比于具体的时间值，我们更关心：随着问题规模的增大，算法的计算复杂性是如何变化的？即以问题规模n为变量的计算复杂性函数$f(n)$的属性。
  - 问题规模的度量与所采用的编码测量（包括描述问题所需的字符集，以及描述方式）有关。

算法的计算复杂性

- 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 e，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。

算法的计算复杂性

- 考虑典型的旅行商问题。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。旅行商问题的任何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。例如：
  C={C1,C2,C3,C4},d(C1,C2)=10,d(C1,C3)=5,d(C1,C4)=9,d(C2,C3)=6,d(C2,C4) = 9和d(C3,C4)=3就构成一个具体例子，而这个例子的一个解是排序<C1,C2,C4,C3>因为四个城市的这个排序所对应的路线是所有可能旅行路线中长度最小的，为27。



算法的计算复杂性

若用字母表

{C, [,], /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

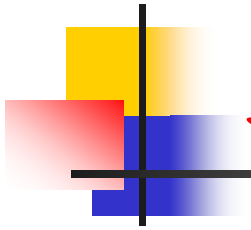
中的字符表示旅行商问题的具体例子，则前面例子可以用字符串

C[1],C[2],C[3],C[4]//10/5/9//6/9//3

来描述。其相应的输入长度为32，从而称这个例子的大小为32。

算法的计算复杂性

- 定义：算法的时间复杂度
- 对某一问题Π和任一可能的输入长度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。
- 对于不同的算法，就有着不同的时间复杂性函数。复杂性函数的不同变化方式反映了算法的好坏程度。例如，时间复杂性函数关于输入长度的增长速度就是判别一个算法时间效率的主要指标。



算法的计算复杂性

- 在计算机科学中存在这样一个一般的约定：仅当算法的时间复杂性函数随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。
  - 多项式时间算法（polynomial time algorithm）
  - 指数时间算法（exponential time algorithm）

算法的计算复杂性

- 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。这类算法的时间复杂性函数典型的例子有2^n,$n!$,n^n,n^{lbn}等。

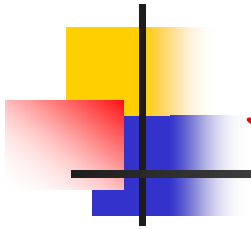
算法的计算复杂性

- 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$c'g(n) \leqslant f(n) \leqslant cg(n)$，则记 $f(n) = O(g(n))$。
- 利用$O(*)$可以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。

算法的计算复杂性

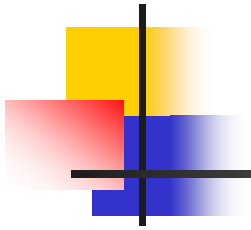
表 5.1　不同复杂性函数增长速度的比较与技术进步的影响

时间复杂性函数	不同 n 值对应的函数值或其近似值			一天内所能解例子的规模	计算机速度提高10倍后，一天内所能解例子的规模
n	10	100	1000	10^{12}	10^{13}
n^3	10^3	10^6	10^9	10^4	2.15×10^4
$n\text{lb}n$	33	664	9966	0.948×10^{11}	0.87×10^{12}
$n^{\text{lb}n}$	2099	1.93×10^{13}	7.89×10^{29}	79	95
2^n	1024	1.27×10^{30}	1.05×10^{301}	40	43
$n!$	3,628,800	10^{158}	4×10^{2567}	14	15



算法的计算复杂性

- 总结：
  - 首先，我们所定义的时间复杂性为最坏情形度量。
  - 其次，在考虑算法的复杂性时，我们通常只关心算法在问题例子的规模n充分大时的表现。
  - 第三，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，本质上并不依赖特定的编码和具体的计算模型。



问题的复杂性分类

- 为了能在统一的框架下刻画、研究任何问题的可行解与求解它的复杂程度，并起到便于描述算法的定义以及相关的理论结果，需要对问题的定义进行进一步抽象与统一化。
- 目前所广泛采用的描述问题的方法主要有两种：一是将任一问题转化为所谓的可行性检验问题（feasibility problem），二是把问题转述为判定问题（decision problem）。

问题的复杂性分类

- “判定问题”是答案只有是与非两种可能的问题。一个判定问题Π可简单地由其所有例子的集合D_{Π}和答案为“是”的例子所构成的子集$Y_{\Pi} \subseteq D_{\Pi}$来组成。
- 为了反映实际问题所具有的特性，通常所采用的标准描述方法由两部分组成。“例子”：用诸如集合、图、函数等各种描述分量来刻画判定问题的一般性例子；“问题”：陈述基于一般性例子所提出的一个“是/非”提问。
- 实际中几乎所有问题都可直接或间接地转述为判定问题。

问题的复杂性分类

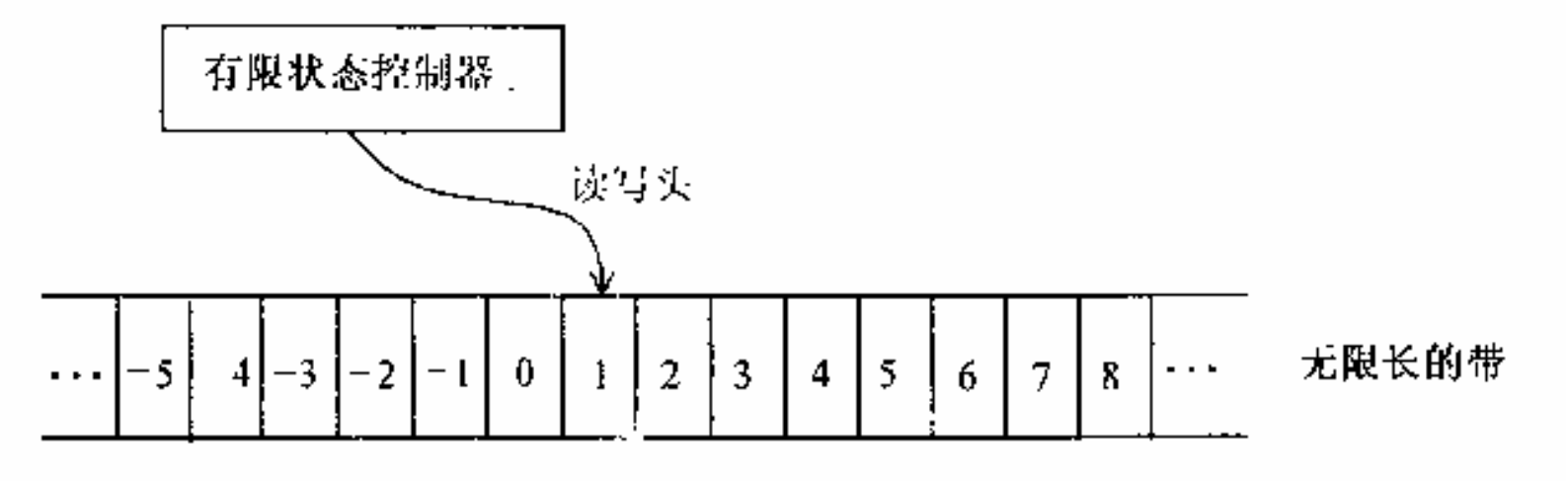
- 语言（language）：对于字符的任一有限集合Σ，用Σ^*表示由Σ中的字符所组成的所有有限长度字符串的集合。若L为Σ^*的一个子集，则称L为字符集Σ上的一个语言。例如，{01,001,111,1010110}就是字符集Σ = {0, 1}上的一个语言。
- 判定问题与语言之间的对应关系是通过编码策略e来实现的。定义“语言”：

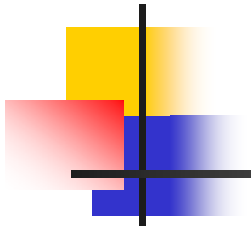
$$L[\Pi,e] = \{ x \in \Sigma^* : \Sigma\text{为}e\text{所使用的字符集、而}x\text{为某个例子在}e\text{下的编码} \}$$

- 如果一个结论对语言$L[\Pi,e]$成立，我们就说它在编码策略e之下对问题Π成立。

问题的复杂性分类

- 确定性单带图灵机（Deterministic one-tape Turing Machine——DTM）：一个DTM由一个有限状态控制器、一个读写头和一条双向的、具有无限多个带格的线性带组成。





问题的复杂性分类

一个DTM程序（program）应详细规定下列信息:

（1）带中所有字符的一个有限集合Γ，它应包括输入字符表子集$\Sigma \subset \Gamma$和一个特别的空白符号$b \in \Gamma - \Sigma$。

（2）一个有限状态集Q，它包括初始状态q_0和两个特有的停机状态q_Y和q_N。

（3）一个转移函数 $\delta : (Q - \{q_Y, q_N\}) \times \Gamma \to Q \times \Gamma \times \{l, r\}$

可以设计一个DTM程序来完成任何可在某一通常计算机上实现的任一计算。

问题的复杂性分类

- 称一个带有输入字符表Σ的DTM程序M接受$x \in \Sigma^*$，当且仅当它作用于输入 x 时停机于状态q_Y 。
- 只有当一个DTM程序对定义于其输入字符表上的所有可能字符串均停机时，我们才称其为一个算法。相应地，称一个DTM程序M在编码策略e之下求解判定问题 Π ，如果M对定义于其输入字符表上的所有输入字符串均停机且有$L_M = L[\Pi, e]$。

问题的复杂性分类

- 一个DTM程序M对于输入x的计算所用的时间定义为该程序从开始直至进入停机状态为止所运行的步数。
- 时间复杂性函数的定义：对于一个对所有输入 $x\in\Sigma^*$ 均停机的DTM程序M，定义其时间复杂性函数

  $T_M: Z^+\rightarrow Z^+$为

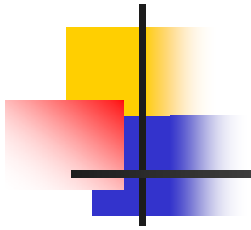
  $T_M(n)$ = max{m: 存在一个$x\in\Sigma^*$，$|x|=n$，使得M对输入x的计算需要时间m}。

问题的复杂性分类

- 若存在一个多项式$p(n)$，使得对所有的$n \in Z^+$，有$T_M(n) \le p(n)$，则称程序M为一个多项式时间DTM程序。
- 定义P类语言（问题）：

P = { L:存在一个多项式时间DTM程序M，使得L= L_M }

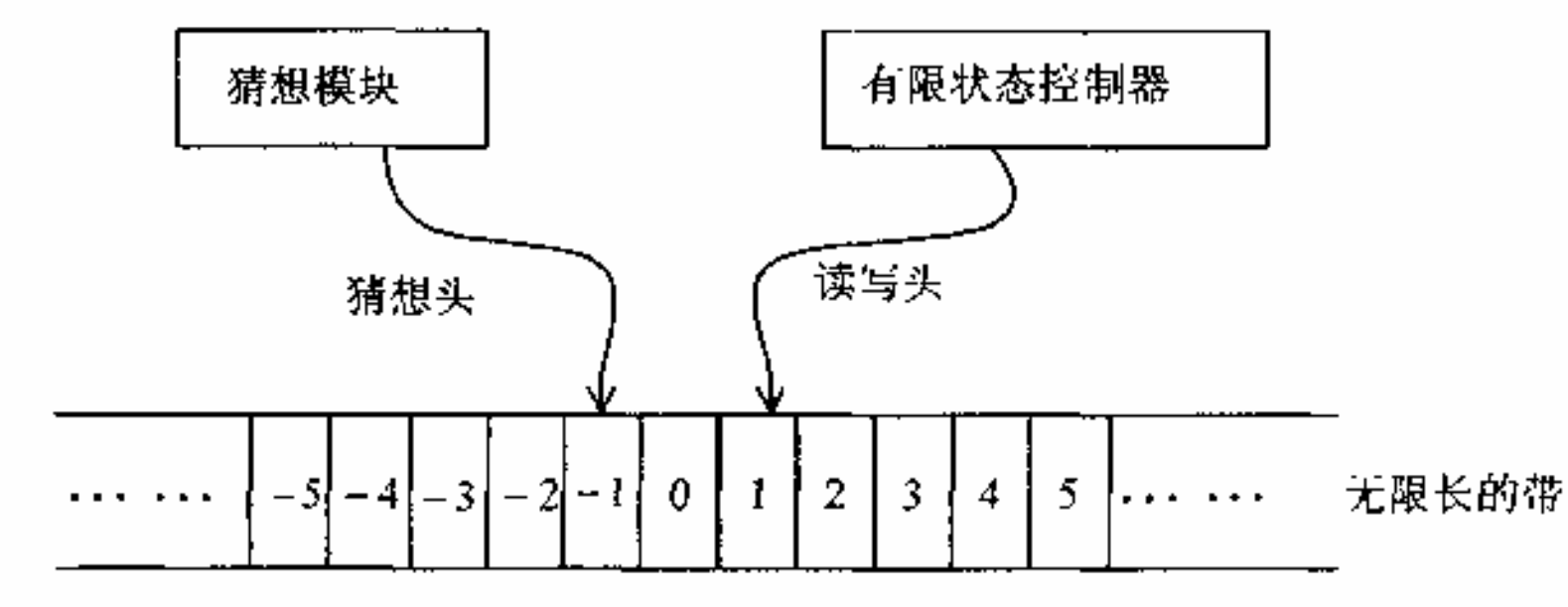
- 若存在一个多项式时间DTM程序，它在编码策略e之下求解Π，即$L[\Pi,e] \in P$，则称该判定问题Π属于P。



问题的复杂性分类

- 非确定性单带图灵机（Non-Deterministic one-tape Turing Machine——NDTM，又称非确定性图灵机）完全是一种假想的机器，通常有两种方法来描述它：多值模型和假想模块模型。
- 用假想模块模型，NDTM可描述成：除多了一个猜想模块外，它与DTM有着完全相同的结构，而这个猜想模块带有自己的仅可对条带写的猜想头，它提供了写下猜想的办法，并仅用于此目的。

问题的复杂性分类



- 对于一个输入$x \in \Sigma^*$，NDTM程序的计算分为两个不同的阶段：猜想阶段和检验阶段。

问题的复杂性分类

- 一般来说，对于一个给定的输入字符串x，NDTM程序M将会做无限多个可能的计算，对每一个可能猜想串都有一个相应的计算，如果这些计算中至少有一个为可接受的计算，则称NDTM程序M接受x，相应地，M所识别的语言定义为：

$$L_M = \{x \in \Sigma^* : M\text{接受}x\}$$

问题的复杂性分类

- 定义一个NDTM程序M接受$x \in L_M$所需地时间为：在M对x的所有可接受计算中，程序从一开始直到进入停机状态q_Y为止，在猜想和检验阶段所进行的步数的最大值。
- M的时间复杂性函数 $T_M: Z^+ \to Z^+$ 则定义为

$T_M(n) = max[\{1\} \cup \{m:$ 存在一个长度为n的$x \in L_M$，使得M要接受它所需的时间为$m\}]$

问题的复杂性分类

- 若存在一个多项式 $p(n)$，使得对所有的$n \geq 1$，有$T_M(n) \leq p(n)$，则称NDTM程序M为一个多项式时间NDTM程序。
- 定义NP类语言（问题）：
- **NP = {L: 存在一个多项式时间NDTM程序M，使得$L_M=L$}**
- 称判定问题Π在编码策略e之下属于NP，若$L[\Pi, e] \in$NP。相应的求解算法称多项式时间不确定算法。

问题的复杂性分类

- 若$\Pi \in NP$，则存在一个多项式 p 使得Π可以用一个时间复杂性为 $O(2^{p(n)})$ 的确定性算法来求解。
- 在非确定性图灵机上时间复杂性为 $q(n)$ 的判定问题和在确定性图灵机上时间复杂性为 $q(n)k^{q(n)}$ 的问题相当，因此，直观上我们有理由认为非确定性图灵机要比确定性图灵机的处理速度快得多，应能有效求解后者所不能有效解决的许多复杂问题。
- $P \subseteq NP$

问题的复杂性分类

- 所谓从一个语言 $L_1 \subseteq \Sigma_1^*$到另一个语言$L_2 \subseteq \Sigma_2^*$的多项式变换是指满足下面两个条件的函数f: $\Sigma_1^* \to \Sigma_2^*$:

  (1) 存在计算f的一个多项式时间DTM程序。

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

用记号$L_1 \propto L_2$表示存在一个从L_1到L_2的多项式变换。

- 称一个语言L（判定问题Π）为NP完全的(NPC)，如果$L \in$NP($\Pi \in$NP)，且对所有别的语言$L' \in$NP($\Pi' \in$NP)，均有$L' \propto L$($\Pi' \propto \Pi$)。

问题的复杂性分类

- 关系$\propto$在这些语言（或判定问题）的等价类上建立了一个偏序（$L_1 \propto L_2$ 意味着L_1比L_2不难）。
- P类问题组成了这一偏序下“最小”的等价类，从而可以看作是由计算上“最容易”的语言（判定问题）所构成。
- 由NP完全的语言（判定问题）所形成的另一个等价类则包含了NP类中那些“最困难”的语言（问题）。

问题的复杂性分类

- 设Π_1和Π_2都是判定问题。我们说Π_1在多项式时间内可图灵规约[简称（图灵）规约]为Π_2，如果存在Π_1的一个算法A_1，它多次调用求解Π_2的算法A_2作为其子程序，并且若假设每次调用该子程序A_2均需用单位时间，则A_1为一个多项式时间的算法。并把A_1叫做从Π_1到Π_2的多项式时间规约，记为$\propto_T$。

问题的复杂性分类

- 如果Π_1可图灵规约到Π_2，那么Π_2是多项式时间可解蕴涵Π_1是多项式时间可解的，从而表明Π_2至少和Π_1一样的难。
- 定义：对于判定问题Π，若存在一个NP完全的问题Π'，使得$\Pi' \propto_T \Pi$，则称问题Π是NP困难的（NP-hard）

经典最优化方法的一般结构

- 通常采用迭代方法
- 基本思想：给定解空间中的一个初始点$x_0 \in R^n$，按照某一迭代规则产生一个点序列$\{x_k\}$，使得当$\{x_k\}$是有穷点列时，其最后一个点是最优化模型问题的最优解。
- 一个好的算法应具备的典型特征为：迭代点x_k能稳定地接近局部极小点x^*的邻域，然后迅速收敛于x^*。当给定的某种收敛准则满足时，迭代即终止。

经典最优化方法的一般结构

- 设x_k为第k次迭代点，d_k为第k次搜索方向，α_k为第k次步长因子，则第k次迭代为

$$x_{k+1} = x_k + \alpha_k d_k$$

- 从这个迭代格式可以看出，不同的步长因子α_k和不同的搜索方向d_k构成了不同的方法。
- 在经典最优化方法中，搜索方向d_k是 f 在x_k点处的下降方向，即d_k满足 $\nabla f(x_k)^T d_k < 0$

  或者 $f(x_k + \alpha_k d_k) < f(x_k)$

经典最优化方法的一般结构

最优化方法的基本结构为:

(1) 给定初始点x_0,

(a) 确定搜索方向d_k，即依照一定规则，构造 f 在x_k点处的下降方向作为搜索方向.

(b) 确定步长因子α_k，使目标函数值有某种意义的下降

(c) 令 $x_{k+1} = x_k + \alpha_k d_k$

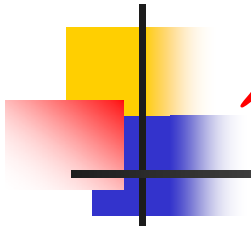
(2) 若x_{k+1}满足某种终止条件，则停止迭代，得到近似最优解x_{k+1}。否则，重复以上步骤.

优化算法及其分类

- 经典算法。包括线性规划、动态规划、整数规划和分枝定界等运筹学中的传统算法，其算法计算复杂性一般很大．只适于求解小规模问题，在工程中往往不实用。
- 构造型算法。用构造的方法快速建立问题的解，通常算法的优化质量差，难以满足工程需要。譬如，调度问题中的典型构造型方法有：**Johnson法、Palmer法、Gupta法、CDS法、Daunenbring的快速接近法、NEH法等。**

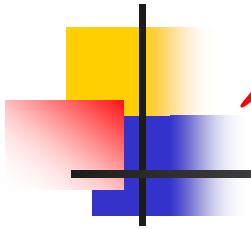
优化算法及其分类

- 改进型算法，或称邻域搜索算法。从任一解出发，对其邻域的不断搜索和当前解的替换来实现优化。根据搜索行为，它又可分为局部搜索法和指导性搜索法。
  - 局部搜索法。以局部优化策略在当前解的邻域中贪婪搜索，如只接受优于当前解的状态作为下一当前解的爬山法；接受当前解邻域中的最好解作为下一当前解的最陡下降法等。
  - 指导性搜索法。利用一些指导规则来指导整个解空间中优良解的探索，如SA、GA、EP、ES和TS等。



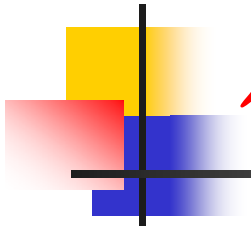
优化算法及其分类

- 基于系统动态演化的方法。将优化过程转化为系统动态的演化过程，基于系统动态的演化来实现优化，如神经网络和混沌搜索等。
- 混合型算法。指上述各算法从结构或操作上相混合而产生的各类算法。



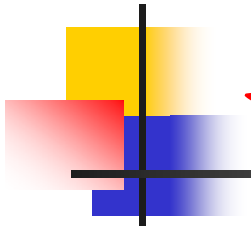
优化算法及其分类

- 枚举法
- 确定性算法
  - 各类数学规划算法，如单纯形法，分支定界法，牛顿法，。。。。
- 随机算法
  - 各类自然计算方法，如模拟退火法，禁忌搜索法，演化算法，。。。。



优化算法及其分类

- 用户往往面临两种选择:
  - 寻找全局最优解，不管耗费多少计算资源;
  - 在用户许可的资源（主要是时间资源）消耗范围内，寻找尽可能“好”的解;
- 在严重缺乏先验知识的情况下，如何尽量快速地找到问题的“好”的解?



一些基本策略

- 贪心策略（Greedy Strategy）
- 局部搜索（Local Search）
  - 如何得到“好”的初始可行解？
  - 如何选择一个“好”的邻域结构或其它定义方式？
  - 确定某一搜索它的方法。
- 群体策略（Population-based Search）
- 分而治之（Divide and Conquer）
- 在搜索过程中学习（Learning during Search）