CHAPTER 1

第1章 概论

本书讨论的现代优化计算方法，主要包括禁忌搜索(tabu search)算法、模拟退火(simulated annealing)算法、遗传算法(genetic algorithms)、蚁群优化算法(ant colony optimization algorithm)和人工神经网络(artificial neural networks)算法等. 这些算法借助现代计算机作为工具，对复杂的组合最优化问题的求解具有普遍适用性，自20世纪80年代以来得到了快速发展和广泛应用，所以我们称它们为现代优化计算方法. 这类算法中的每一个算法都以人类、生物的行为方式或物质的运动形态为背景，经过数学抽象建立算法模型，通过计算机的计算来求解组合最优化问题，因此这些算法也被称之为元启发式(meta-heuristics)算法.

现代优化算法的发展非常迅速，例如，在1996年，Osman在本章参考文献[1]中分类罗列了1400篇相关文章. 因此，这些算法同人工智能、计算机科学和运筹学相融合就不足为奇了. 现代优化计算方法涉及人工智能、分子运动、遗传学、动物学、神经系统和统计力学等学科的概念和理论，以模型的抽象为其关键点，以数学为其理论基础. 随着人们对客观世界认识的发展及计算机技术的提高，现代优化计算方法所涵盖的内容将不断扩大.

本章主要介绍组合最优化问题、计算复杂性和启发式算法的概念.

1.1 组合最优化问题

组合最优化(combinatorial optimization)是通过数学方法的研究去寻找离散事件的最优编排、分组、次序或筛选等，是运筹学(operations

research)中的一个经典和重要的分支,所研究的问题涉及信息技术、经济管理、工业工程、交通运输、通信网络等诸多领域.

我们知道,最优化问题的数学模型的一般描述是

$$\min_{x\in F} f(\boldsymbol{x}),$$

其中,$\boldsymbol{x}$ 为决策变量,$f(\boldsymbol{x})$为目标函数,F 为可行域,F 中的任何一个元素称为该问题的一个可行解,满足 $f(\boldsymbol{x}^*)=\min\{f(\boldsymbol{x})\mid \boldsymbol{x}\in F\}$的可行解 $\boldsymbol{x}^*$ 称为该问题的最优解,对应的目标函数值 $f(\boldsymbol{x}^*)$称为最优值.

组合最优化问题的特点在于,可行域 F 表示的是有限个点组成的集合.通常情况下,可行域 F 可以表示为$\{\boldsymbol{x}\mid \boldsymbol{x}\in D,\ g(\boldsymbol{x})\geqslant 0\}$,所以组合最优化问题的一般形式为

$$\begin{aligned}&\min\ f(\boldsymbol{x})\\ &\text{s.t.}\ \ g(\boldsymbol{x})\geqslant 0,\\ &\qquad \boldsymbol{x}\in D.\end{aligned}$$

因此,一个组合最优化问题可用三参数(D,F,f)表示,其中 D 表示决策变量的定义域,$F=\{\boldsymbol{x}\mid \boldsymbol{x}\in D,\ g(\boldsymbol{x})\geqslant 0\}$表示可行解区域(可行域),$f$ 表示目标函数.组合最优化的特点是可行解的集合 F 为有限点集,决策变量的定义域 D 通常也是有限点集.由直观可知,只要将 D 中有限个点逐一判别是否满足 $g(\boldsymbol{x})$的约束和比较目标值的大小,该问题的最优解一定存在且可以得到(除非可行域为空集).因为现实生活中的大量优化问题是从有限个状态中选取最好的,所以大量的实际优化问题是组合最优化问题.

例 1.1.1 0-1 背包问题(knapsack problem).

设有一个容积为 b 的背包,n 个尺寸分别为 $a_i(i=1,2,\cdots,n)$,价值分别为 $c_i(i=1,2,\cdots,n)$的物品,如何装包以使装入物品的总价值最大?这个问题称为 0-1 背包问题,可用数学模型表示为

$$\max\ \sum_{i=1}^{n} c_i x_i \tag{1.1.1}$$

$$\text{s.t.}\ \ \sum_{i=1}^{n} a_i x_i \leqslant b, \tag{1.1.2}$$

$$x_i\in\{0,1\},\quad i=1,\cdots,n, \tag{1.1.3}$$

其中目标(1.1.1)欲使包内所装物品的价值最大,式(1.1.2)为包的能力限制,式(1.1.3)表示 x_i 为二进制变量,$x_i=1$ 表示装第 i 个物品,$x_i=0$ 表示不装.此时若采用三参数表示法,$D=\{0,1\}^n$,F 为 D 中满足式(1.1.2)的解集合(可行解域),f 为目标函数(1.1.1).

□

例 1.1.2 旅行商问题(traveling salesman problem,TSP).

一个商人欲到 n 个城市推销商品,每两个城市 i 和 j 之间的距离为 d_{ij},如何选择一条

道路使得商人每个城市走一遍后回到起点且所走路径最短？TSP 还可以细分为对称和非对称距离两大类问题. 当 $d_{ij}=d_{ji}$，$\forall i,j$ 时，称为对称距离 TSP，否则为非对称距离 TSP. 对一般的 TSP，一种数学模型描述为

$$\min \sum_{i\neq j} d_{ij}x_{ij} \tag{1.1.4}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij}=1, \quad i=1,2,\cdots,n, \tag{1.1.5}$$

$$\sum_{i=1}^{n} x_{ij}=1, \quad j=1,2,\cdots,n, \tag{1.1.6}$$

$$\sum_{i,j\in S} x_{ij}\leqslant |S|-1, \quad 2\leqslant |S|\leqslant n-2, \quad S\subset\{1,2,\cdots,n\}, \tag{1.1.7}$$

$$x_{ij}\in\{0,1\}, \quad i,j=1,\cdots,n, i\neq j. \tag{1.1.8}$$

以上是基于图论的数学模型，其中式(1.1.8)中的决策变量 $x_{ij}=1$ 表示商人行走的路线包含从城市 i 到城市 j 的路径，$x_{ij}=0$ 表示商人没有选择走这条路. $i\neq j$ 的约束可以减少变量的个数，使得共有 $n\times(n-1)$ 个决策变量. 目标(1.1.4)要求距离之和最小. 式(1.1.5)要求商人从城市 i 恰好出来一次，式(1.1.6)要求商人恰好走入城市 j 一次. 式(1.1.5)和式(1.1.6)合起来表示每个城市恰好经过一次. 仅有约束(1.1.5)和(1.1.6)无法避免子回路的产生(一条回路是由 $k(1\leqslant k\leqslant n)$ 个城市和 k 条弧组成的一个环)，因此，式(1.1.7)约束旅行商在任何一个城市真子集中不形成回路，其中 $|S|$ 表示集合 S 中元素个数. 此时若采用三参数表示法，$D=\{0,1\}^{n\times(n-1)}$，F 为 D 中满足式(1.1.5)，式(1.1.6)和式(1.1.7)的解集合(可行域)，f 为目标函数(1.1.4). □

上面两个问题都是组合最优化中的经典问题. 它们的共性是，通过对实际问题的数学简化，读者容易理解这些问题；决策变量的定义域和可行解集合都是有限的，在问题的规模较小时，通过枚举很容易得到最优解.

例 1.1.3　整数线性规划(integer linear programming).

$$\text{(IP)} \qquad \begin{aligned} &\min \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}\\ &\text{s.t.}\ \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b},\\ &\boldsymbol{x}\geqslant\boldsymbol{0}, \quad \boldsymbol{x}\in\mathbb{Z}^n, \end{aligned}$$

其中，$\boldsymbol{c}$ 为 n 维列向量，$\boldsymbol{A}$ 为 $m\times n$ 矩阵，$\boldsymbol{b}$ 为 m 维列向量，$\boldsymbol{x}$ 为 n 维决策变量，$\mathbb{Z}^n$ 表示 n 维整数向量的集合. 模型中的系数 $\boldsymbol{A}$、$\boldsymbol{b}$ 和 $\boldsymbol{c}$ 的元素都是整数. □

例 1.1.1 和例 1.1.2 的数学模型都具有以上(IP)的形式. 整数线性规划同线性规划形式上非常相似，不同之处是决策变量部分或全部取整数.

例 1.1.4　装箱(bin packing)问题.

设有 n 个一维尺寸不超过 1 的物品集合$\{a_1,a_2,\cdots,a_n\}$，如何以个数最少的一维尺寸

为1的箱子装进这 n 个物品？该问题为(一维)装箱问题. □

例 1.1.5 约束机器排序(capacitated machine scheduling)问题[2].

n 个加工量为 $\{d_i \mid i=1,2,\cdots,n\}$ 的产品在一台机器上加工，机器在第 t 个时段的工作能力为 c_t，求出所有产品得以加工的最少时段数. 它的数学模型可以表示为

$$\min\ T \tag{1.1.9}$$

$$\text{s.t.}\ \sum_{t=1}^{T} x_{it} = 1,\quad i = 1,2,\cdots,n, \tag{1.1.10}$$

$$\sum_{i=1}^{n} d_i x_{it} \leqslant c_t,\quad t = 1,2,\cdots,T, \tag{1.1.11}$$

$$x_{it} \in \{0,1\},\quad i = 1,2,\cdots,n, t = 1,2,\cdots,T, \tag{1.1.12}$$

其中，x_{it}，T 为决策变量，$x_{it}=1$ 表示第 t 时段加工产品 i. 式(1.1.9)要求加工所用的时段数最少，式(1.1.10)表示产品 i 一定在某一个时段加工，式(1.1.11)表示每个时段的加工量不超过能力的限制. □

整数规划模型的建立有助于对问题的理解和采用已有的如分支定界、割平面等算法求最优解，也利于松弛变量的整数约束条件得到问题的下界或上界. 组合优化问题通常可以用整数规划模型的形式表示(如例1.1.1和例1.1.2等)，但有些组合最优化问题用整数规划模型表示比较复杂和不易被理解，不如问题的直接叙述容易理解，如例1.1.2、例1.1.4和例1.1.5. 同时，由于组合最优化问题的复杂性，很多快速的算法只给出问题的可行解. 因此，根据对解的不同精度要求和分析的需要，有大量的组合最优化问题是通过文字语言叙述的，不一定强求给出整数规划模型.

1.2 计算复杂性的概念

计算复杂性的概念是为评估算法的计算耗用时间和解的偏离程度等指标而提出的. 它以目前二进制计算机中的储存和计算为基础，以理论的形式系统描述. 这一套理论产生于20世纪70年代，目前仍然是评估算法性能的基础.

首先，以简单的实例来理解算法的计算耗用时间. 由组合最优化问题的定义，每一个组合最优化问题都可以通过枚举的方法求得最优解. 枚举是以时间为代价的，有的枚举时间还可以接受，有的则不可能接受. 对例1.1.2的非对称距离TSP问题，可用另一个方法来表示它的可行解：n 个城市的一个排列表示商人按这个排列顺序推销并返回起点. 若固定一个城市为起终点，则需要 $(n-1)!$ 次枚举. 以计算机1s可以完成24个城市所有路径枚举(固定起点后，实际上是余下的23个城市的所有排列)为单位，则25个城市的计算时间为：以第1个城市为起点，第2个到达城市有可能是第2、第3、……、或第25个城市.

决定前两个城市的顺序后，余下是 23 个城市的所有排列，枚举这 23 个城市的排列需要 1s，所以，25 个城市的枚举需要 24s. 类似地归纳，城市数同计算时间的关系如表 1.2.1 所示.

表 1.2.1　枚举时城市数与计算时间的关系

城市数	24	25	26	27	28	29	30	31
计算时间	1s	24s	10min	4.3h	约 4.9d	约 136.5d	10.8a	约 325a

通过表 1.2.1 可以看出，随着城市数的增多，计算时间增加得非常快，当城市数增加到 30 时，计算时间约 10.8a，已无法接受.

上面的分析同样带来疑问：计算时间会不会随计算机的计算速度不同而不同呢？回答显然是肯定的. 因此需要理论上规范计算时间这个概念. 针对一个如 TSP 的问题，人们可以给出以上的枚举算法，这个算法并没有限定城市的个数和城市间的距离. 但对具体的计算机，算法是以计算机语言实现的，它只能对城市数和城市间距离确定的问题给出答案. 于是，我们先从计算复杂性中的“问题”和“实例”等基本概念开始讨论.

问题(problem)是一个抽象的模型或概念，是需要回答的一般性提问，通常含有若干个满足一定条件的参数. 问题通过下面的描述给定：(1)描述所有参数的特性；(2)描述答案满足的条件. 如 TSP 是一个问题，由例 1.1.2 的文字或模型(1.1.4)～(1.1.8)表示. 该问题的参数是城市数和城市间的距离，而答案(经过所有城市并回到出发城市的旅行路线)满足的条件是，该旅行路线在所有可能的旅行路线中是最短的. 例 1.1.1～例 1.1.5 都是问题.

一个算法常常是针对一个问题来设计的，如 TSP 的枚举算法可以求解任何一个 TSP 的最优解. 而若用计算机求解，则必须对问题中的参数赋予具体值，如 TSP 中的城市数和城市间的距离，问题中的参数赋予了具体值的例子称为问题的实例(instance). 一个问题通过它的所有实例表现，实例是问题的表现形式，问题是实例的抽象(集合). 例 1.1.1 的背包问题，当物品的个数、包的容积、每个物品的价值和尺寸给定具体数值后，决定它的一个实例. 在给出具体的城市数和城市间的距离后，确定了例 1.1.2 的 TSP 问题的一个实例.

为了在计算机上实现算法对实例的求解，首先必须将实例输入并存储在计算机中. 具体地说，需要将实例中的参数的具体数值存储. 这些数值必然占据计算机的存储空间. 以二进制编码存储的计算机为例，一个正整数 $x\in[2^s,2^{s+1})$ 的二进制展开

$$x=a_s2^s+a_{s-1}2^{s-1}+\cdots+a_1\times 2+a_0\quad(a_s=1,a_i\in\{0,1\},i=0,1,\cdots,s-1),$$

与系数 $(a_sa_{s-1}\cdots a_1a_0)$ 一一对应. 该数可由 $(a_sa_{s-1}\cdots a_1a_0)$ 表示. 这个二进制数的位数是

$s+1$,也就在计算机中占据 $s+1$ 位的空间. 由

$$2^s \leqslant 2^{\log_2 x} = x < 2^{\log_2(x+1)} = x+1 \leqslant 2^{s+1} \leqslant 2^{1+\log_2 x},$$

则

$$\log_2(x+1) \leqslant s+1 \leqslant 1+\log_2 x.$$

因为

$$1+\log_2 x - \log_2(x+1) < 1, \quad \forall x \geqslant 1,$$

所以任意正整数 x 占用位数为

$$l(x) = \lceil \log_2(x+1) \rceil, \tag{1.2.1}$$

其中$\lceil x \rceil$表示不小于 x 的最小整数,且

$$\log_2 x < \lceil \log_2(x+1) \rceil \leqslant 1+\log_2 x. \tag{1.2.2}$$

需要注意的是整数 0,1 的二进制位数都是 1,所以特别定义$\lceil \log_2 1 \rceil = 1$. 再考虑一个符号位和一个数据分隔位,上面的二进制表示方法和输入规模的结论(1.2.1)可以推广到任何整数 x,任何一个整数 x 的输入规模为(包含一个符号位和一个数据分隔位)

$$l(x) = \lceil \log_2(|x|+1) \rceil + 2. \tag{1.2.3}$$

一个数在计算机中存储时占据的位数称为这个数的规模或编码长度(size),由式(1.2.3)决定. 用 $l(I)$ 表示实例 I 的规模. 一个实例的规模定义为这个实例所有参数数值的规模之和(包括不同数据之间的特殊分隔符). 常规理解的计算机只能用有限位存储一个实数,因此,在计算复杂性概念的介绍中,我们限定只考虑有理数,或者也可以限定只考虑整数.

例 1.2.1 TSP 实例的规模.

对例 1.1.2 的 TSP 问题,它的任何一个实例由城市数 n 和城市间的距离 $D=\{d_{ij} \mid 1 \leqslant i,j \leqslant n, i \neq j\}$ 确定. 于是,TSP 的任何一个实例 I 的规模(考虑符号和数据分隔位)为

$$l(I) = \lceil \log_2(n+1) \rceil + 2 + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \left\{ \lceil \log_2(|d_{ij}|+1) \rceil + 2 \right\},$$

其中,$\lceil \log_2(n+1) \rceil + 2$ 为 n 的存储空间. 根据式(1.2.2),可以得到

$$2n(n-1) + 2 + \log_2 |P| < l(I) \leqslant 3n(n-1) + 3 + \log_2 |P|, \tag{1.2.4}$$

其中, $P = n \prod_{d_{ij} \neq 0, i \neq j} d_{ij}$ □

例 1.2.2 整数线性规划的规模.

例 1.1.3 的整数线性规划问题的实例需要存储系数 $m,n,\boldsymbol{c},\boldsymbol{A}$ 和 $\boldsymbol{b}$,其中,$\boldsymbol{c}$ 为 n 维列向量,$\boldsymbol{A}$ 为 $m \times n$ 矩阵,$\boldsymbol{b}$ 为 m 维列向量. 在所有系数为有理数的假设前提下,可以通过通分,使得 $\boldsymbol{A},\boldsymbol{b},\boldsymbol{c}$ 各分量为整数. 由式(1.2.3),$\boldsymbol{c} = (c_1, c_2, \cdots, c_n)^{\mathrm{T}}$,$\boldsymbol{A} = (a_{ij})_{m \times n}$,$\boldsymbol{b} =$

$(b_1,b_2,\cdots,b_m)^{\mathrm{T}}$的二进制规模(考虑符号、数据分隔位)分别是

$$L_1 = 2n + \sum_{i=1}^{n} \lceil \log_2(|c_i|+1) \rceil,$$

$$L_2 = 2mn + \sum_{i=1}^{m}\sum_{j=1}^{n} \lceil \log_2(|a_{ij}|+1) \rceil,$$

$$L_3 = 2m + \sum_{i=1}^{m} \lceil \log_2(|b_i|+1) \rceil,$$

占用的总位数为 $l(I)=4+\lceil \log_2(m+1)\rceil+\lceil \log_2(n+1)\rceil+L_1+L_2+L_3$. 再根据式(1.2.2)可以得到

$$2(mn+m+n+2)+\log_2|P| < l(I) \leqslant 3(mn+m+n+2)+\log_2|P|, \tag{1.2.5}$$

其中,P 为 mn 与 $\boldsymbol{A}$,$\boldsymbol{b}$ 和 $\boldsymbol{c}$ 中所有非零数的乘积. □

继续研究表 1.2.1 的有关 TSP 的计算量. 可以肯定,随着计算机技术的发展,计算机计算速度会大幅度提高,因此可以计算更大规模的 TSP 实例. 但是作为衡量计算量的一个标准,我们希望给出的衡量计算量的方法不能就计算机的不同而不同,而只与实例本身有关.

一个算法解一个实例的计算量定义为,算法求解中的加、减、乘、除、比较、读和写磁盘等基本运算的总次数. 从定义中可以看出,一旦实例和算法给定,这个量是不变的(假定这里不考虑带有非确定性的算法). 如表 1.2.1 中提及的枚举算法,当城市数为 n 且第一个城市为始终点时,计算一条路径$(1,i_2,\cdots,i_n)$长度的基本运算为两两城市间距离的 n 个求和. 因为有$(n-1)!$ 条路径,枚举所有路径进行$(n-1)!$ 次比较,可以得到最优路径. 这个枚举算法的基本计算(加法和比较)总次数为$(n-1)!n+(n-1)!=(n-1)!(n+1)$.

记问题的一个实例为 I,实例规模为 $l(I)$,算法 A 在求解 I 时的计算量(基本计算总次数)为 $C_A(I)$. 当存在一个函数 $g(x)$和一个常数 α,使得对于该问题任意的实例 I 均满足

$$C_A(I) \leqslant \alpha g(l(I)), \tag{1.2.6}$$

则用 $C_A(I)=O(g(l(I)))$表示. 它的含义是:算法解实例 I 的计算总次数$C_A(I)$是实例规模 $l(I)$的一个函数,这个函数被另一个函数 $g(l(I))$控制. 函数 $g(x)$的函数特性决定了算法的性能.

定义 1.2.1　假设问题和解决该问题的一个算法已经给定,若存在多项式函数$g(\boldsymbol{x})$,使得式(1.2.6)对给定问题的所有实例成立,我们称该算法为解决对应问题的多项式时间算法.

根据定义 1.2.1,一个多项式时间算法考虑了所有的基本运算,因此,它的输出结果

的规模一定也与实例输入规模呈多项式关系.

为了方便,当不存在多项式函数 $g(x)$ 使得式(1.2.6)成立时,称相应的算法是非多项式时间算法或指数时间算法.相比较而言,随着变量的增加,多项式函数增长的速度比指数函数增长的速度要慢得多.例如,随着 n(大于2的整数)的增加,n^k($k>0$ 为整数)的增长速度远比 a^n($a>1$ 为实数)增长的速度慢.若一个多项式时间算法的计算总次数为 $C_1=2^{10}n^2$,而另一个非多项式时间算法的计算次数为 $C_2=2^n$.当 n 比较小时,多项式时间算法的优越性并不明显.如当 $n=10$ 时,$C_1/C_2=10^2$,多项式时间算法的计算次数是非多项式时间算法的 10^2 倍;但当 $n=2^5$ 时,$C_1/C_2=2^{-12}$,非多项式时间算法的计算次数上升非常快,是多项式时间算法的计算次数的 2^{12} 倍.

从式(1.2.6)和定义1.2.1中可以发现,我们以算法计算总次数的上界和实例的规模之间存在的函数关系 $g(x)$ 来评价算法的优劣.由于计算量根据计算总次数的上界来估计,观察式(1.2.4)和式(1.2.5)的上下界估计,发现任何一个实例规模的上下界偏差不超过这个实例的规模,且上下界与实例规模在同一个数量级上,于是,实际进行算法复杂性分析时,可用实例规模的下界作为一个实例规模的估计,计算总次数的上界作为计算量的估计.同样从上面的例子看出,无论是否考虑数据的分隔位或符号,实例规模都在一个数量级上,因此,可以简单地用参数的个数和所有非零数值乘积的绝对值的对数来估计.

重述一个观点,我们构造算法的目的是能够解决问题的所有实例而不单单是问题的一个实例.根据这个观点,人们逐步形成了计算复杂性的概念,主要包括算法复杂性分析和问题复杂性分析.

算法复杂性分析的研究目的是对算法进行评价.对于解决一个问题的算法,如何评估这个算法的性能?算法复杂性分析是通过式(1.2.6)在最坏实例的条件下,确定是否存在多项式函数 $g(x)$.

由实例的输入规模和算法在求解实例时的计算量可以找到它们之间的关系.通过这个关系可以衡量算法的好坏.对例1.1.2的TSP问题,任何一个实例的规模由式(1.2.4)确定,且实例规模的上下界在同一个数量级,取下界为 $2(n-1)^2+2+\log_2|P|$.而枚举算法的计算总次数为 $(n-1)!(n+1)$.此时发现,TSP实例的二进制规模是 n 和 P 的多项式函数.枚举算法的计算量是 n 的阶乘函数,无法将计算量同实例的规模建立多项式关系.

下面再以线性规划中的单纯形算法为例进行说明.先估计线性规划任何一个实例的二进制编码规模.

例1.2.3 线性规划问题(linear programming).

$$
\text{(LP)}\qquad
\begin{aligned}
&\min\ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}\\
&\text{s.t. } \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b},\\
&\boldsymbol{x}\geqslant\boldsymbol{0},\quad \boldsymbol{x}\in\mathbb{R}^n,
\end{aligned}
$$

其中，$\boldsymbol{c}$ 为 n 维列向量，$\boldsymbol{A}$ 为 $m\times n$ 矩阵，$\boldsymbol{b}$ 为 m 维列向量，$\boldsymbol{A},\boldsymbol{b},\boldsymbol{c}$ 各分量为整数，$\boldsymbol{x}$ 为 n 维决策变量. 与例 1.2.2 相同的讨论，它的每个实例 I 的规模 $l(I)$同样满足关系式(1.2.5).

单纯形算法是解决线性规划问题的主要经典算法之一，对于大多数实例，它的计算效果非常好，这可从实际工程、管理人员的大量实际应用得以证实. 也可以说，对线性规划问题的很多实例，单纯形算法的计算量是实例规模的多项式函数，但也存在极端的情形. Klee 和 Minty 在本章参考文献[3]中给出下面这样一个实例：

$$\begin{aligned}
&\min\ (-x_n)\\
&\text{s.t. } 4x_1-4r_1=1,\\
&\quad x_1+s_1=1,\\
&\quad 4x_j-x_{j-1}-4r_j=0,\quad j=2,3,\cdots,n,\\
&\quad 4x_j+x_{j-1}+4s_j=4,\quad j=2,3,\cdots,n,\\
&\quad x_j,r_j,s_j\geqslant 0,\quad j=1,2,\cdots,n.
\end{aligned}\tag{1.2.7}$$

本章参考文献[3]中证明单纯形算法求解线性规划(1.2.7)的迭代步数是 2^n-1. 由于实例(1.2.7)共有 $2n$ 个约束和 $3n$ 个变量，且系数的最大值为 4，P 是实例(1.2.7)中所有非零数的乘积，所以 $\log_2|P|\leqslant(6n^2+1)\log_2 4$. 由式(1.2.5)知，$l(I)=O(n^2)$.

由此可见，对于线性规划的单纯形算法，不存在多项式函数 $g(x)$和常数 α，使得对问题的任意一个实例 I，都有式(1.2.6)成立. 否则的话，不妨假设该多项式函数 $g(x)$为非负增函数，则

$$2^n-1\leqslant C_A(I)\leqslant \alpha g(l(I))\leqslant \alpha g(O(n^2))=O(g_1(n)),$$

其中 $g_1(x)$也是一个多项式函数. 显然，上面的式子在 n 充分大时是不可能成立的.

因此，单纯形算法不是求解线性规划问题的多项式时间算法. □

计算复杂性理论的另一个研究方向是问题的复杂性分析，即对问题按计算复杂性归类. 可以如下定义多项式问题.

定义 1.2.2　对于给定的一个优化问题，若存在一个求最优解的算法、一个多项式函数 $g(x)$和常数 α，使得式(1.2.6)对该问题的所有实例成立，则称给定的优化问题是多项式时间可解问题，或简称多项式问题. 所有多项式问题的集合记为 P(polynomial).

以例 1.2.3 的线性规划问题为例，已经得到单纯形算法不是线性规划问题的多项式时间算法的结论，但这并不能说明线性规划问题不属于多项式问题. Khachian 在本章参考文献[4]中成功地构造了椭球算法并证明了其算法是线性规划问题的多项式时间算法. 因此，线性规划问题是多项式问题.

并不是所有的组合最优化问题都找到了求最优解的多项式时间算法. 也就是说，还不能肯定一些组合最优化问题是否属于 P. 经过几代数学家的努力，他们研究、整理了一类

难以求解的组合最优化问题,迄今为止,这些问题还没有一个有能求最优解的多项式时间算法.这一类组合最优化问题归为所谓的NP完全(NP-complete)或NP难(NP-hard)问题.受人类认识能力的限制,目前人们只能假设这一类难的组合最优化问题不存在求最优解的多项式时间算法.本章的例1.1.1～例1.1.5都属于这类难解问题.有关计算复杂性的更多概念将在1.5节和1.6节进一步讨论,感兴趣的读者可以继续阅读这一部分或本章参考文献[5].

正因为一些组合最优化问题还没有找到求最优解的多项式时间算法,而这些组合最优化问题又有非常强的实际应用背景,人们才不得不尝试用一些并不一定可以求得最优解的算法,通常称为启发式算法,来求解组合最优化问题.

1.3 邻域的概念

在距离空间中,通常的邻域定义是以一点为中心的一个球体.光滑函数极值的数值求解中,邻域是一个非常重要的概念,求解中通过寻求上升或下降的方向,以点到点的迭代,达到函数值的上升或下降.在组合最优化中,欧氏距离的概念通常不再适用,但是在一点附近搜索另一个下降的点是组合最优化数值求解的基本思想.因此,需要重新定义邻域的概念.

定义 1.3.1 对于组合最优化问题(D,F,f),D上的一点到D的子集的一个映射

$$N:\boldsymbol{x}\in D\to N(\boldsymbol{x})\in 2^D,$$

且$\boldsymbol{x}\in N(\boldsymbol{x})$,称为一个邻域映射,其中$2^D$表示$D$的所有子集组成的集合.$N(\boldsymbol{x})$称为$\boldsymbol{x}$的邻域,$\boldsymbol{y}\in N(\boldsymbol{x})$称为$\boldsymbol{x}$的一个邻居.增加$\boldsymbol{x}\in N(\boldsymbol{x})$的限制是为了同距离空间的邻域定义更为相似.

例 1.3.1 例1.1.2已给出TSP的一种数学模型,由模型$D=\{\boldsymbol{x}\mid\boldsymbol{x}\in\{0,1\}^{n(n-1)}\}$,可以定义它的一种邻域为

$$N(\boldsymbol{x})=\Big\{\boldsymbol{y}\mid\boldsymbol{y}-\boldsymbol{x}\mid=\sum_{i,j}\mid y_{ij}-x_{ij}\mid\leqslant k,\boldsymbol{y}\in D\Big\},\quad k\text{为一个正整数.}$$

这个邻域定义使得$\boldsymbol{x}$最多有k个位置的值可以发生变化,$\boldsymbol{x}$的邻居有$1+C_{n(n-1)}^1+C_{n(n-1)}^2+\cdots+C_{n(n-1)}^k$个(注意它们不一定都是可行解). □

例 1.3.2 TSP问题解的另一种表示法为$D=\{\boldsymbol{S}=(i_1,i_2,\cdots,i_n)\mid i_1,i_2,\cdots,i_n$是$1,2,\cdots,n$的排列$\}$.本章参考文献[6]中定义它的邻域映射为2-opt,即$\boldsymbol{S}$中的两个元素进行对换,$N(\boldsymbol{S})$中共包含$\boldsymbol{S}$的C_n^2个邻居和$\boldsymbol{S}$本身.如四个城市的TSP实例,当$\boldsymbol{S}=(1,2,3,4)$时,$N(\boldsymbol{S})=\{(1,2,3,4),(2,1,3,4),(3,2,1,4),(4,2,3,1),(1,3,2,4),(1,4,3,2),(1,2,4,3)\}$. □

2-opt的定义思想可以推广到k-opt$(k\geqslant 2)$,即对$\boldsymbol{S}$中的k个元素按一定的规则