高等学校物流工程与物流管理专业系列规划教材

现代优化技术

XIANDAI YOUHUA JISHU

靳志宏　计明军　编著

大连海事大学出版社
Dalian Maritime University Press

第1章 优化问题与优化技术

1.1 优化问题

1.1.1 现实中的优化问题

现代化管理以及工程技术领域遇到的众多决策问题最终都可以归结为优化问题。物流工程与管理以及交通运输规划与管理等领域的若干现实优化问题举例如下：

例 1-1 设施选址问题(facility location problem)

设有 n 个客户，第 j 个客户的地理位置坐标为 (a_j,b_j)，该客户对某种货物的需求量为 q_j，$j=1,\cdots,n$。现规划设立 m 个配送网点，第 i 个网点的容量为 c_i，$i=1,\cdots,m$，试确定网点的地理位置，合理规划物流网络的结构与布局，使物流成本最低或效率最高等。如图 1-1 所示。

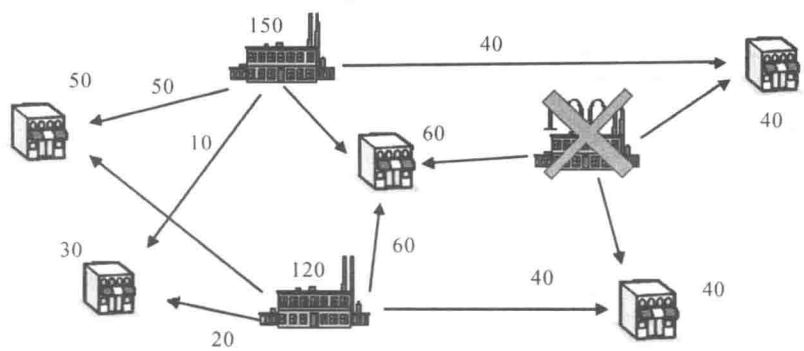

图 1-1 设施选址问题

设第 i 个网点的地理位置坐标为 (x_i,y_i)，$i=1,\cdots,m$；第 i 个网点为第 j 个客户配送货物量为 z_{ij}，$i=1,\cdots,m$，$j=1,\cdots,n$。则该问题用数学模型表示为：

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} z_{ij}\sqrt{(x_i-a_j)^2+(y_i-b_j)^2} \tag{1-1}$$

$$\text{s.t.} \sum_{j=1}^{n} z_{ij} \leqslant c_i, i=1,\cdots,m \tag{1-2}$$

$$\sum_{i=1}^{m} z_{ij} = q_j, j=1,\cdots,n \tag{1-3}$$

$$z_{ij} \geqslant 0, i=1,\cdots,m; j=1,\cdots,n \tag{1-4}$$

式(1-1)要求所建立网点到各客户的运输量与运输距离的积(吨公里)之和最小化;式(1-2)表示第i个网点的配送量不能超过其容量;式(1-3)表示第j个客户得到的供应量应等于其需求量;式(1-4)为非负限制。

例 1-2 运输规划问题(transportation planning problem)

某物流网络系统由多个批发商向多个零售商提供多种货物所组成。各批发商的供货能力及其地理位置、各零售商的需求量及其地理位置已知,试确定各批发商每种货物的供货范围和相应的供货量,使整个物流网络运输成本最低。如图 1-2 所示。

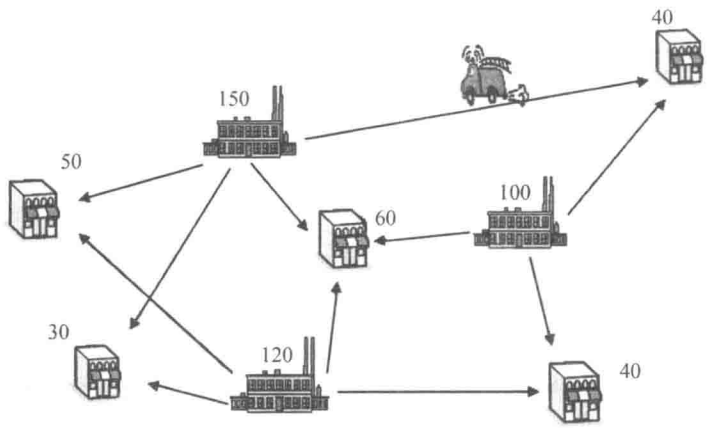

图 1-2 运输规划问题

设第i个供货商,其第k种货物的供应能力为$A_{ik}, i = 1,\cdots,m, k = 1,\cdots,l$;第$j$个零售商,其对第$k$种货物的需求量为$B_{jk}, j = 1,\cdots,n, k = 1,\cdots,l$;第$i$个供货商到第$j$个零售商的单位运输成本为$C_{ij}$,决策变量为第$i$个供货商向第$j$个零售商供应第$k$种货物数量为$x_{ijk}$。则该决策问题可以用数学模型表示为:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} \sum_{k=1}^{l} x_{ijk} \tag{1-5}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ijk} \leqslant A_{ik}, i = 1,\cdots,m; k = 1,\cdots,l \tag{1-6}$$

$$\sum_{i=1}^{m} x_{ijk} = B_{jk}, j = 1,\cdots,n; k = 1,\cdots,l \tag{1-7}$$

$$x_{ijk} \geqslant 0, i = 1,\cdots,m; j = 1,\cdots,n; k = 1,\cdots,l \tag{1-8}$$

式(1-5)显示优化目标为使总成本达到最小;式(1-6)表示每个批发商对每种货物的供货量不能超过其供货能力;式(1-7)表示零售商j对货物k的需求量必须得到满足;式(1-8)为非负约束。

例 1-3 背包问题(knapsack problem)

设有一个容积(或承重量)为b的背包,n个体积(重量)分别为$a_i(i = 1,\cdots,n)$、价值分别为$c_i(i = 1,\cdots,n)$的物品,如何装包才能使背包内的价值最大?如图 1-3 所示。

该问题用数学模型可以表示为:

$$\max \sum_{i=1}^{n} c_i x_i \tag{1-9}$$

$$\text{s.t.} \sum_{i=1}^{n} a_i x_i \leqslant b \tag{1-10}$$

$$x_i \in \{0,1\}, i = 1,\cdots,n \tag{1-11}$$

其中,式(1-9)欲使背包内所装载的物品价值最大,式(1-10)为背包能力限制,式(1-11)表示 x_i 为二进制决策变量,$x_i = 1$ 表示装载第 i 个物品,$x_i = 0$ 表示不装载第 i 个物品。

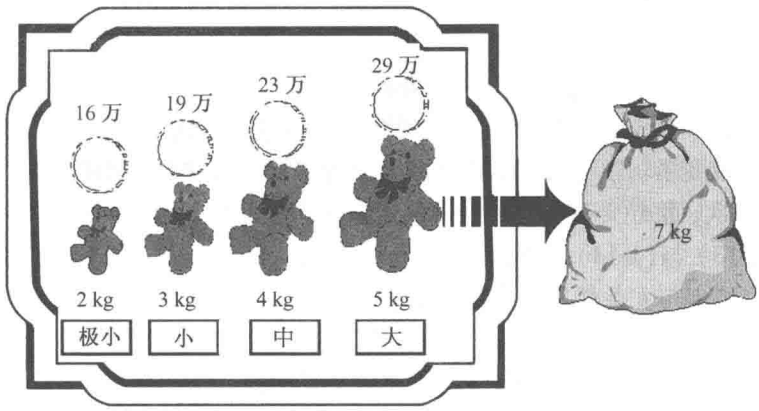

图 1-3 背包问题

背包问题在集装箱装箱等物流领域具有广泛的应用。

例 1-4 旅行商问题(traveling salesperson problem)

一个推销员要到 n 个城市推销商品,每两个城市 i,j 间的距离为 d_{ij},如何选择一巡回路径使得推销员在每个城市推销一周后回到出发点所行走的距离最短。如图 1-4 所示。

图 1-4 旅行商问题

该问题可以用多种模型加以描述,其中一种数学模型如下:

$$\min \sum_{i \neq j} d_{ij} x_{ij} \tag{1-12}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1, i = 1, \cdots, n \tag{1-13}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, \cdots, n \tag{1-14}$$

$$\sum_{i,j \in s} x_{ij} \leqslant |s| - 1, 2 \leqslant s \leqslant n-2, s \subset \{1, \cdots, n\} \tag{1-15}$$

$$x_{ij} \in \{0,1\}, i, j = 1, \cdots, n, i \neq j \tag{1-16}$$

式(1-16)中的决策变量 $x_{ij} = 1$ 表示推销员行走的路径包含从城市 i 到城市 j 的路段；$x_{ij} = 0$ 表示推销员没有行走从城市 i 到城市 j 的路段。$i \neq j$ 这一约束用于减少变量的个数，即共有 $n(n-1)$ 个决策变量。式(1-12)表示目标函数为路径距离之和最小化。式(1-13)要求推销员从城市 i 走出一次，而式(1-14)要求推销员走进城市 j 一次，两者表示每个城市仅路过一次，但仅仅满足式(1-13)与(1-14)却不能避免子回路(subtour)的产生。子回路是指一条回路由 $k(1 \leqslant k < n)$ 个城市与 k 条弧所组成的闭循环路。式(1-15)保证推销员的巡回路径不会形成子回路，其中 $|s|$ 表示集合 s 中组成元素的个数。

旅行商问题在生产物流以及运输与配送领域具有十分广泛的应用。

1.1.2 优化问题分类

上述优化问题尽管表现形式不同，但都由如下一些基本的要素所组成：

(1)决策变量(decision variable)

决策变量通常为优化问题要求解的未知量，一般可以用 n 维向量表示 $x = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$，当对其赋值后得到该优化问题的一个解。

(2)目标函数(objective function)

目标函数通常为该问题所要优化(最大化或最小化)的目标的数学表达式，为决策变量的函数式，一般可以简记为 $f(x)$。

(3)约束条件(constraints)

约束条件即对决策变量的限制条件 $g(x) \leqslant 0$，是 $x = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$ 允许的取值范围。其表现形式通常为一组关于 x 的等式或不等式，分别称为等式约束与不等式约束。

具有上述三个基本要素，优化问题可以表述为如下一般形式：

$$\text{Opt. (min or max)} \quad z = f(x) \tag{1-17}$$

$$\text{s.t.} \quad g_j(x) \leqslant 0, j = 1, \cdots, m \tag{1-18}$$

$$x = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}, x \in \Omega$$

1.1.2.1 连续优化(continuous optimization)

在上述一般化表述中，若决策变量 $x = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$ 的所有分量取值均为连续数值，即实数时，优化问题称为连续优化问题。上述现实优化问题的例 1-1 与例 1-2 均属于连续优化问题。工程技术领域的优化问题大都为连续优化问题。

此时，若 $f(x), g_j(x)$ 都是线性函数，称为线性规划(linear programming)；若 $f(x), g_j(x)$ 至少有一个非线性函数，称为非线性规划(nonlinear programming)；若 $f(x)$ 是一个二次函数，而

$g_j(x)$是线性函数,称为二次规划(quadratic programming),是一种特殊的、相对简单的非线性规划。

1.1.2.2 离散优化(discrete optimization or combinatorial optimization)

若决策变量 $x=(x_1,x_2,\cdots,x_n)^T$ 的一个或多个分量只取离散数值时,优化问题称为离散优化,或称为组合优化。上述现实优化问题的例1-3与例1-4均属于离散优化问题。管理领域的优化问题,如排序问题、分配问题等,大都为离散优化问题。

当决策变量 $x=(x_1,x_2,\cdots,x_n)^T$ 的一个或多个分量只取整数值时,称为整数规划(integer programming);当决策变量 $x=(x_1,x_2,\cdots,x_n)^T$ 的所有分量只取整数值时,称为纯整数规划(pure integer programming);当决策变量 $x=(x_1,x_2,\cdots,x_n)^T$ 的部分分量只取整数值时,称为混合整数规划(mixed integer programming);特别的,当决策变量 $x=(x_1,x_2,\cdots,x_n)^T$ 的所有分量只取值0或1时,称为0-1规划(zero-one programming)。

一般来说,上述两大类优化问题中,离散优化问题因"组合爆炸"的存在而使计算复杂程度更高,进而比连续优化问题更难以求解。

优化问题除了上述分类方法外,还可以根据其组成三要素的其他特点进行分类。如根据约束条件的特征分为无约束优化问题与有约束优化问题;根据目标函数的特征分为单一目标优化与多目标优化;根据决策变量的时变性分为静态规划与动态规划;根据决策环境分为确定性规划与随机规划;根据优化问题的层次分为单层规划与多层规划等(唐焕文,2003)。

1.2 优化技术

优化技术就是在满足约束条件的可行解集合内寻求全局最优解、局部最优解(近似最优解、满意解)的过程中所应用的理论与方法,涉及运筹学理论与计算机技术等多学科的交叉领域,具体包括数学建模、约束处理、算法设计及程序设计等主要环节。

1.2.1 可行解、局部最优解与全局最优解

在上述优化问题的一般化表述中,同时满足所有约束条件式(1-18)的解 x 称为可行解(feasible solution),否则称为非可行解。如果在某个可行解 $x_{l\text{-}opt}$ 的某个邻域 $N(x_{l\text{-}opt})$ 内,$x_{l\text{-}opt}$ 使目标函数值式(1-17)达到最优,即将可行域限定在 $N(x_{l\text{-}opt})$ 中时的最优解,但不一定是整个可行域 Ω 上的最优解,则称 $x_{l\text{-}opt}$ 为关于邻域 $N(x_{l\text{-}opt})$ 的局部最优解(local optimal solution)。相对于局部最优解,将整个可行域 Ω 上的最优解称为全局最优解 $x_{g\text{-}opt}$(global optimal solution)。

可行解、局部最优解、全局最优解的关系如图1-5所示。

对于大多数现实优化问题,求解全局最优解是非常困难的,甚至常常是不可能的,只能求得其局部最优解(又称为满意解或近似最优解)。基于此,优化技术可分为求解全局最优解的精确解技术以及求解局部最优解的近似解技术。本书重点针对的是后者。

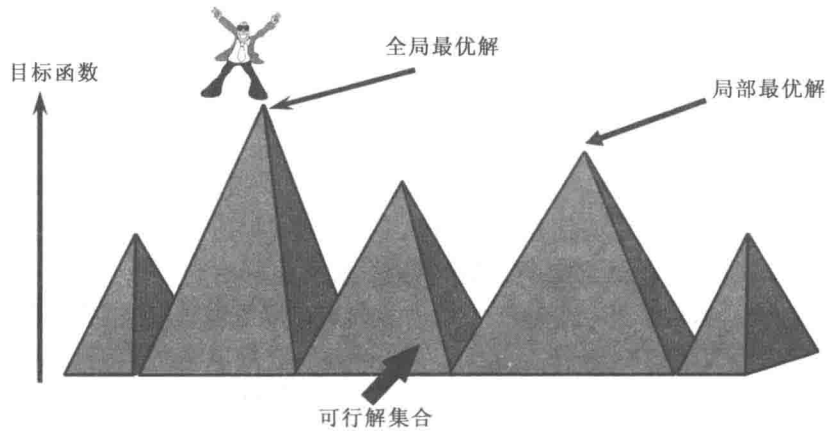

图 1-5　可行解与局部最优解以及全局最优解的关系示意图

1.2.2　数学建模

从上述现实中的优化问题及其数学表现形式可以看出，虽然这些问题各具特色，但是其间都有一个共同点，即针对一个实际问题，通过辨识问题中变量及其相互关系来将现实问题转化为由数学语言描述的问题，这就是数学建模的过程。

数学建模的方法大体上分为两类。其一是机理与机制分析法，即通过分析现实问题的影响因素及其特征、相互关系，挖掘出反映问题本质及其反应机理、作用机制的内部规律，所设置的各类变量及其制约式能够具有现实意义。其二是统计与模拟分析法，即运用数理统计分析或模拟仿真分析，按照一定的规则寻找一个与历史数据拟合得最理想的表达形式。在实际的建模过程中，也常常将上述两类方法结合运用。通过机理与机制分析确定模型结构，用数理统计分析或模拟仿真分析确定模型参数。

数学建模过程主要包括下述几个环节。

1.2.2.1　建模准备

对现实问题建模，必须首先了解其背景、环境，掌握所研究问题的类型、所依据的事实，明确建模的目的，确定数据、资料的来源及其可靠性。收集、整理资料并对其进行数据挖掘等预处理工作。

1.2.2.2　建模假设

一般情况下，现实问题如果不经过简化很难直接转化为数学问题，而任何简化都需要一定的假设条件。建模假设就是要辨识并能够列出与优化问题相关的因素，通过假设把所要研究的实际问题简化、抽象，明确模型中需要考虑的因素，以及它们之间的关系和在问题当中的作用及影响。用数量和参数的形式表示出这些因素及其内在联系。

1.2.2.3　建模分析

根据建模假设分析变量与参数之间的相关关系，利用其间的内在规律，运用数学方面的技

能、技巧来描述,构建各个变量之间的数学结构与定量关系。使用已知数据、观测数据、调查数据以及实际问题的有关背景知识对所建模型的参数给出估计值。在此基础上,还要进行误差分析以及敏感性分析。

1.2.2.4 建模检验

利用已知数据运行所建模型,解释模型的结果或将其与实际观测结果进行分析、比较。如果模型结果的解释与实际情况相符合,表明模型符合实际,可以进一步用其实践。反之,如果模型结果的解释与实际情况不相符合,不能直接应用于所研究的实际问题,此时,需返回到建模准备以及建模假设阶段。也可以通过理论解析,从理论方面探讨解的性质,并对实际问题给出相应的解释。

1.2.3 约束处理

现实优化问题几乎都存在约束,如果违背了这些约束条件就无法得到最优解、甚至可行解。这些约束条件可分为硬约束(hard constraint)与软约束(soft constraint)。所谓硬约束,就是为了找到可行解必须完全满足的条件;所谓软约束,是指希望满足但又不是必须满足的条件。通常,软约束条件都必须进行量化,通过增加惩罚项来度量其违反软约束的程度,并通过某种形式与目标函数结合起来(Michalewicz Z. & Fogel D., 2003)。

在解决现实优化问题时,通常有以下几种约束处理手段:

①在编码过程中保证所构建的解满足约束条件,即通过设计与问题相适应的可行解的表示方式。

②在可行解进化探索过程中,设法保持解的可行性。即使用专门的表示方式和变化算子来保持解集合中个体的可行性。

③拒绝非可行解,即在探索过程中,将产生的候选解与约束条件一一对照,一旦发现违反约束,便将其从候选解集合中剔除。这种做法具有很大的局限性,因为对于很多优化问题而言,更重要的是对非可行解进行改进,而不是简单地拒绝。

④惩罚非可行解,即通过候选解的评价值(目标函数值)对非可行解进行惩罚。需要设计适当的可行解评价函数,以及设计适当的非可行解的惩罚函数来自动降低非可行解的质量。

⑤修补非可行解,即根据问题的特性,将一个非可行解修补为可行解。值得注意的是,这一修补设计具有对问题本身特点的依赖性,对每个特定的问题必须设计不同的修补过程。

⑥转变非可行解为可行解,可以通过改变探索空间的拓扑结构实现这一转变,也可以通过识别可行解与非可行解之间的临界面,借助于适当的惩罚或诱导使探索向希望的方向进行。

1.2.4 算法设计

1.2.4.1 精确解算法与近似解算法

所谓算法就是优化问题的程序化解决方案。这些解决方案本身并不是问题的答案,而是获得答案的一系列指令集合,即一步步求解问题的通用程序。如果存在一个算法,它对某一优

化问题的每一个实例，在有限步骤后一定可以得到该实例的一个答案，那么就称该算法可以求解该优化问题。

对于一个优化问题，如果给定任意一个实例，某算法总能找到一个可行解，那么就称其为该优化问题的近似算法；如果进一步，这个可行解的目标值总等于最优解值，则称其为该问题的精确解算法。

例如，单纯形算法是求解线性规划问题的精确解算法、表上作业法是求解运输问题的精确解算法、匈牙利算法是求解指派问题的精确解算法等。但对于大多数现实优化问题而言，由于：

①简化模型与实际问题的吻合度的差异；

②可行解的探索空间巨大；

③求解时间过长、成本过高；

④实际约束过于苛刻，可行解构造困难；

⑤约束条件以及可行解的动态变化；

⑥解的评价标准的动态变化；等等。

求解精确解或是困难的，甚至是不可能的或是不必要的。尽管管理领域的很多优化问题属于离散的组合优化问题，而组合优化问题的每个实例的可行解都是有穷集合，通过穷举法逐一对比总可以找到最优解，但是由于该算法计算量过大，用于求解现实规模的问题往往是不实用的，这时就需要求助于近似解算法。本书旨在介绍实用的近似算法开发相关技术。

1.2.4.2 算法设计的核心环节

（1）理解优化问题

设计一个解决现实问题的实用算法首先需要做的就是完全理解要优化的问题。实用优化算法是为现实问题设计的，这些实用优化问题尽管表现形式千差万别，但都具有一些重要的特征。根据这些特征可将其大致归为以下几大类型：

①排序问题。即按照升降序排列给定列表中的数据项，相应的算法称为排序算法。

②查找问题。即从给定的集合中寻找一个给定项，相应的算法称为查询算法。

③字符串处理问题。字符串是由字母、数字以及特殊符号构成的，如何在文本中查到一个给定的词即为串匹配问题，相应的算法称为串匹配算法。

④图问题。图是由一些被称为顶点的点所构成的集合，其中某些顶点由一些被称为边的线段相连。相应的算法称为图算法，如遍历算法、最短路算法、图填色算法等。

⑤组合问题。即通过寻求一个组合对象，如一个排列、一个子集等，使这些对象能够满足特定的条件、具有某些希望的特性等。组合数随着问题规模的扩大迅速增加，目前尚无一个通用有效的组合算法。

⑥几何问题。即处理类似于点、线、面、体等集合对象的问题，相应的算法称为几何算法。

⑦数值问题。即具有连续性的数学对象问题，其相应的算法称为连续优化算法。由于大多数数学问题仅能求解近似解，同时也由于实数在计算机内部只能近似表述，因此多为连续近似算法。

根据这些类型的特征，可以有助于理解优化问题的本质，进而设计出相应的优化算法。

(2) 熟悉计算硬件

现行算法的实现要在计算机硬件设备上运行代码,该体系的核心在于随机存储器(random access machine, RAM)。其基本思路是逐条运行代码指令,每次执行一次操作。相应地,设计运行于这种计算机硬件设备上的算法称为顺序算法。与此相对应,另外一类计算机硬件设备可以在同一时间执行多条操作也即并行计算。该种新式计算机硬件已经超出了 RAM 模式的核心假设。能够利用这种计算性能优势的算法称为并行算法。现行算法设计的主流仍为 RAM 模式下的顺序算法,同时也是并行算法的基础。

(3) 选择优化算法

解决现实优化问题需要根据具体问题选择、设计算法。

通过理论解析等得到的精确解算法主要有穷举法、分枝界定法、动态规划法等等;通过数值分析等得到的近似解算法主要包括连续优化近似算法以及离散优化近似算法。连续优化近似算法包括线性搜索算法及利用二次函数逼近的二次终结性算法等。离散优化近似算法包括基于构筑策略的构筑法、基于改善策略的改善法等。

在精确解算法与近似解算法之间做出选择主要考虑精度要求与计算代价(计算成本、计算时间)之间的折中。在近似解算法之间的选择主要根据问题自身的结构与特点等。

(4) 确定数据结构

所谓数据结构就是对优化问题的相关数据项进行组织的特定方案。算法与数据结构是利用计算机求解优化问题的两个核心基础。数据结构是由数据项的性质决定的,而数据项的性质是由优化问题所决定的。主要的数据结构包括:线性数据结构、图结构、树结构、集合结构、字典结构等。有关详细内容将在随后的优化技术基础的相关章节阐述。

(5) 表述优化算法

主要有三种算法表述形式:自然语言、流程图、伪代码。

用自然语言描述算法简单、易理解。但是自然语言固有的不严密性使得简化而又逻辑清晰地描述算法变得十分困难,特别是对于一些复杂的算法描述显得过于累赘。

流程图是一种使用一系列相连的几何图形来描述算法的方式,几何图形内包含了算法步骤的描述信息。如用圆柱表述原始数据、用长方形表示计算操作、用菱形表示判断等。

伪代码是自然语言与类编程语言组成的混合结构。同自然语言相比,伪代码的表述更精确;同编程语言相比,用伪代码生成的算法描述更简洁。通常,用伪代码表述的算法包括一些精炼的、典型的编程语句,如赋值语句"←"、注释语句"//"、条件语句"if"、循环语句"for"等。这些表述方式仅仅是约定俗成,并没有达成广泛的、具有约束力的共识。从形式上看,似乎大家都在使用各自的"方言"来进行各自为政的表述。但由于这些"方言"与"普通话"都很接近,尚不至于产生歧义。

本书的算法描述采用伪代码的形式。

(6) 分析优化算法

算法分析首先要对其正确性进行分析,即要证明对于每一个合法输入,算法都应该会在有限时间内给出一个满足要求的输出。

对于精确解算法,证明算法正确性的一般方法是使用数学归纳法,这是因为算法本身的迭代过程就已经提供了这种证明所需要的一系列步骤。通常,根据一些特定的输入来跟踪算法操作或者根据一些事先就已经知道输出的结果反过来验证等做法很有意义,但并不能最终证

明算法的正确性。相反，如果要证明算法的不正确性，只需要提供一个“反例”就足够了。

对于近似解算法，证明算法正确性就是要表明该算法产生的误差不会超出事先定义的范围。

其次，算法分析要进行其简洁性分析。所谓简洁性不但要求算法描述本身要简单明了，具有一般性，而且还要求其具有很高的效率。这里的效率包括时间效率与空间效率。时间效率又称为时间复杂性，显示算法运行的时间要求，即其运行得有多快；而空间效率又称为空间复杂性，显示算法运行过程中需要多少存储空间。

再次，算法分析还要进行容错性分析。所谓容错性是指输入不正确时，算法能适当地反应，不至于输出错误的或无意义的结果。容错性是反应算法可靠性的主要指标。

算法分析的详细内容将在算法设计与评价的相关章节中阐述。

1.2.5 程序设计

程序设计是算法实现的途径，即为算法写代码。程序设计是将一个算法正确地编写成计算机程序。程序设计需要掌握至少一门计算机程序设计语言，以及必要的程序设计方法与技巧，这样才能如实反映算法，避免算法走样。

由算法转换后的程序是否正确需要测试与调试。对于实用优化问题而言，对程序的验证主要还是依赖测试。程序测试既是一门科学，也是一门技术，其内容已经超出了本书的范围，请参阅相关专业书籍。

第 2 章 优化技术基础

2.1 最优化理论

2.1.1 凸集及其性质

从几何图形上来看，单个的点，一条直线，两维平面上的圆形、矩形、三角形，三维空间的球体、圆锥体，等等，都是凸向外的图形，这些图形都具有一个共同的特征，即连接图形中的任意两点的直线段都会落在该图形之内。这样的多维空间就是一个凸集。其精确的定义如下所述。

设 D 是 n 维欧氏空间 E^n 的一个子域（$D \subset E^n$），x_1, x_2 是 D 中任意两个点（$x_1, x_2 \in D$），如果连接 x_1, x_2 的直线段上的所有点，即对于 $0 \leqslant \lambda \leqslant 1$ 的一切 λ，总有

$$\lambda x_1 + (1 - \lambda) x_2 \in D \tag{2-1}$$

成立，则称 D 为一个凸集。

凸集具有下列一些基本性质：

① 整个欧氏空间 E^n 是凸集；空集 ϕ 也为凸集。

② 如 D 为凸集，λ 为实数，则集合 $\lambda D = \{x_2 \mid x_2 = \lambda x_1, x_1 \in D\}$ 也为凸集。

③ 如 D_1, D_2 为凸集，则集合 $D_1 + D_2 = \{x \mid x = x_1 + x_2, x_1 \in D_1, x_2 \in D_2\}$ 仍为凸集。

④ 任何一组凸集的交集仍为凸集。

根据上述基本性质，可以推导出下列在优化问题中常常用到的基本性质：

⑤ 线性约束条件组成的可行域 $F = \{x \mid Ax \leqslant B, x \geqslant 0\}$ 是凸集。

⑥ 欧氏空间 E^n 中的超平面 $H = \{x \mid A^{\mathrm{T}}x = B, x \in E^n\}$ 是凸集。

⑦ 以 x_0 为中心，ε 为半径的开球所组成的 x_0 的邻域 $N(x_0, \varepsilon) = \{x \mid \|x - x_0\| < \varepsilon\}$ 也是凸集。

2.1.2 凸组合及其性质

设向量 $\{x_i\}, i = 1, \cdots, n$，如有实数 $\lambda_i \geqslant 0$，且 $\sum_{i=1}^{n} \lambda_i = 1$，则称 $\sum_{i=1}^{n} \lambda_i x_i$ 为向量 $\{x_i\}$ 的一个凸组合。

凸集与凸组合之间的联系如下：

① 如果点 $x_i \in D, i = 1, \cdots, n$ 的任意凸组合仍包含在 D 中，则 D 一定为凸集。

② 设 x_0 为凸集 D 中的一点，如果不存在 D 中的相异点 x_1 和 x_2，以及某一实数 $\lambda \in (0,1)$，使得 $x_0 = \lambda x_1 + (1-\lambda)x_2$，则称 x_0 是 D 的极点。有界闭凸集中的每一个非极点必定是其极点的凸组合。

2.1.3 凸函数及其性质

凸(convex)函数在最优化理论中起着非常重要的作用。

设 $f(x)$ 是定义在非空凸集 $D \subset E^n$ 上的函数，若对于任意 $x_1, x_2 \in D$，不等式

$$f(\lambda x_1 + (1-\lambda)x_2) \leqslant \lambda f(x_1) + (1-\lambda)f(x_2) \tag{2-2}$$

对于 $0 \leqslant \lambda \leqslant 1$ 的一切 λ 都成立，则称 $f(x)$ 为 D 上的凸函数。

若对于任意 $x_1, x_2 \in D$ 不等式

$$f(\lambda x_1 + (1-\lambda)x_2) < \lambda f(x_1) + (1-\lambda)f(x_2) \tag{2-3}$$

对于 $0 \leqslant \lambda \leqslant 1$ 的一切 λ 都成立，则称 $f(x)$ 为 D 上的严格凸函数。

凸函数具有下列基本性质：

① 如果 $f_1(x), f_2(x)$ 是凸集 $D \subset E^n$ 上的凸函数，则 $f_1(x) + f_2(x)$ 也是 D 上的凸函数。

② 如果 $f(x)$ 是凸集 $D \subset E^n$ 上的凸函数，则对于任意常数 $\alpha > 0$，函数 $\alpha f(x)$ 也是 D 上的凸函数。

③ 如果 $f_1(x), f_2(x), \cdots, f_n(x)$ 是凸集 $D \subset E^n$ 上的凸函数，$\lambda_i \geqslant 0, i = 1, \cdots, n$，则非负线性组合 $\sum_{i=1}^{n} \lambda_i f_i(x)$ 也是 D 上的凸函数。

④ 如果 $f(x)$ 是凸集 $D \subset E^n$ 上的一阶可微分函数，则 $f(x)$ 是 D 上的凸函数的充分必要条件为，对任意 $x_1, x_2 \in D$，都有

$$f(x_2) \geqslant f(x_1) + (\nabla f(x))^{\mathrm{T}}(x_2 - x_1) \tag{2-4}$$

式中，$\nabla f(x)$ 为 $f(x)$ 的梯度向量。即

$\nabla f(x) = (\partial f_{x_1}, \partial f_{x_2}, \cdots, \partial f_{x_n})^{\mathrm{T}}$。

⑤ 如果 $f(x)$ 是凸集 $D \subset E^n$ 上的二阶可微分函数，则 $f(x)$ 是 D 上的凸函数的充分必要条件为，$f(x)$ 的 Hesse 矩阵

$$H(x) = \nabla^2 f(x) = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nn} \end{pmatrix} \tag{2-5}$$

在 D 上是半正定的。

式中，$f_{ij} = \dfrac{\partial^2 f(x)}{\partial x_i \partial x_j}$。

凸集与凸函数之间的联系如下：

① 如果 $f(x)$ 是凸集 $D \subset E^n$ 上的凸函数，则对于任意实数 r，水平集 $D_r = \{x \mid x \in D, f(x) \leqslant r\}$ 是一个凸集。

② 如果 $f_1(x), f_2(x), \cdots, f_n(x)$ 是凸集 $D \subset E^n$ 上的凸函数，$r_i (i = 1, \cdots, n)$ 为实常数，则 D 中同时满足 $f_i(x) \leqslant r_i (i = 1, \cdots, n)$ 的点所构成的集合为凸集。

上述凸函数的条件弱化后可以得到广义的凸函数，包括拟凸（quasi convex）函数和伪凸（pseudo convex）函数。当现实优化问题的目标函数或约束条件具备了某些广义凸函数条件时，寻找全局最优解变得较为简单。

设 $f(x)$ 是定义在凸集 $D \subset E^n$ 上的函数，若对于任意两点 $x_1, x_2 \in D$ 以及 $0 \leqslant \lambda \leqslant 1$，均有不等式

$$f(\lambda x_1 + (1-\lambda)x_2) \leqslant \max\{f(x_1), f(x_2)\} \tag{2-6}$$

成立，则称 $f(x)$ 为 D 上的拟凸函数。

设 $f(x)$ 是凸集 $D \subset E^n$ 上的一阶连续可微分函数，则对于任意相异的两点 $x_1, x_2 \in D$，如果满足

$$(\nabla f(x_1))^{\mathrm{T}}(x_2 - x_1) \geqslant 0 \tag{2-7}$$

就必有

$$f(x_2) \geqslant f(x_1) \tag{2-8}$$

成立，则称 $f(x)$ 为 D 上的伪凸函数。

相应地，拟凸函数、伪凸函数与严格拟凸函数、严格伪凸函数的关系类似于凸函数与严格凸函数的关系。

此外，广义凸函数也有类似上述提到的凸函数的一些基本性质。

2.1.4 凸规划及其性质

凸规划与上述凸集、凸组合、凸函数具有密切关联。

不失一般性，优化问题可以简单表述为

$$\begin{cases} \min f(x) \\ \text{s.t. } g_i(x) \leqslant 0, i = 1, \cdots, n \end{cases} \tag{2-9}$$

在问题(2-9)中，若 $f(x)$ 与 $g_i(x)(i = 1, \cdots, n)$ 均为可行解集合 R 上的凸函数，这样的问题称为凸规划问题。

凸规划问题(2-9)具有下列基本性质：

① 凸规划问题的可行解集合（可行域）R 是凸集。

② 凸规划问题的最优解集 R^* 一定是凸集。

③ 凸规划问题的任一局部最优解即为全局最优解。

④ 当凸规划问题的目标函数 $f(x)$ 是严格凸函数时，凸规划问题的最优解是唯一的。

⑤ 设可行域 R 为凸集，$f(x)$ 是 R 上的拟凸函数，则最优解集合 R^* 也为凸集。

⑥ 设可行域 R 为凸集，$f(x)$ 是 R 上的严格拟凸函数，则其局部最优解即为全局最优解。

⑦ 设可行域 R 为凸集，$f(x)$ 是 R 上的伪凸函数，若 $\nabla f(x^*) = 0$，则 x^* 是全局最优解。

⑧ 设可行域 R 为凸集，$f(x)$ 是 R 上的严格伪凸函数，若 $\nabla f(x^*) = 0$，则 x^* 是唯一的全局最优解。

上述性质的证明限于篇幅，证明从略。有兴趣的读者请查阅最优化理论的相关书籍（吴祈宗，2003；邢文训、谢金星，1999；阳明盛、罗长童，2006）。

上述凸规划理论构成了利用现代优化技术求解现实优化问题，可以得到局部最优解（近似解），甚至全局最优解的理论基础。

凸规划理论与连续的数学规划以及离散的组合优化理论共同构筑了现代优化技术的数学方面的基础。

2.2 计算复杂性理论

2.2.1 组合优化与组合爆炸

优化问题包括连续的数学规划与离散的组合优化问题。由于连续优化问题的探索空间十分巨大,利用现代优化技术求解现实的优化问题常常可以将其进行离散化处理,使其转化为组合优化问题。同时,现实生产与生活中的大量优化问题是从有限个离散状态中选取最优的,所以大量的实际优化问题属于组合优化问题。

因此,这里仅讨论组合优化问题与计算的复杂性。

2.2.1.1 组合优化

一个组合优化问题 π 由三个基本要素 (D,F,f) 组成。其中,

①D 表示决策变量的定义域,其每个决策变量的有限个取值的不同组合形式构成了组合优化问题的不同实例 I,所有这些实例构成了实例集合 $S(I)$。

② 对于每一个实例 I,存在一个有穷的可行解集合 $F(I)$。

③ 对于每一个实例 I 和每一可行解 $\sigma \in F(I)$,其目标函数都可以被赋予一个实数值 $f(I,\sigma)$。如果组合优化问题 π 为求极大(小)值问题,则实例 I 的最优解 σ^* 是这样一个可行解,即对于 $\forall \sigma \in F(I)$,都满足 $f(I,\sigma^*) \geqslant f(I,\sigma)$(对于极大值问题则为 $f(I,\sigma^*) \leqslant f(I,\sigma)$)。

对于一个组合优化问题 π,给定任意一个实例 I,如果能设计一个解法程序 P 使其总能够找到该问题的一个可行解 $\sigma \in F(I)$,则称 P 为 π 的近似算法;进一步,如果总能够找到该问题的一个最优解 σ^*,则称 P 为 π 的精确算法。

组合优化问题的最大特点是决策变量取值是离散的、有限的;可行解集合同样也是有限的。因此,从理论上讲,只要将这些有限的点逐一判断是否满足约束条件和比较目标函数值的大小,该问题的精确解算法一定存在并可以找到。但是,由于现实优化问题的规模都比较大、约束条件较为复杂等原因,在可以容忍的有限时间及费用范围内,通过精确算法寻找最优解十分困难,不得已退而求其次,需要采用近似算法求其局部最优解。其根本原因就在于下面提到的组合爆炸(柳浦睦憲と茨木俊秀,2001)。

2.2.1.2 组合爆炸

组合爆炸(combinatorial explosion)是指随着优化问题规模的不断增大,决策变量取值的不同组合量、可行解数量以及寻找最优解时需要考虑的组合量也会迅速大幅度增加,且往往是以指数形式增加,最终导致无法从中找到最优解。

例 1-3 的背包问题,如果共有 n 个物品,每个物品都有被装进背包与不被装入背包 2 种可能性,因此,该问题共有 2^n 种可行解。

例 1-4 的旅行商问题,对于有 n 个城市的旅行商问题,若出发点城市固定的话,旅行商的下

一个可供选择的要访问的城市有 $n-1$ 个，接下来的可供选择的城市有 $n-2$ 个……以此类推，该问题共有 $(n-1)!$ 种可行解。

上述问题是典型的组合优化问题，当问题规模（如上述问题的物品数或城市数 n）增大时，其可行解的数量以不可思议的速度膨胀。

表 2-1 显示了例 1-3 背包问题可行解数量 2^n 随着问题规模 n 的增大而迅速膨胀的情形。

表 2-1 背包问题组合爆炸的数值解

问题规模 n	2^n 的组合数
0	1
1	2
2	4
8	256
17	131072
26	67108864
35	34359738368
44	17592186044416
53	9007199254740992
62	4611686018427387904
71	2361183241434822606848
80	1208925819614629174706176

伴随着组合优化问题的组合爆炸，利用计算机通过枚举法求得最优解的计算量也迅速攀升，计算时间随之迅速延长。

表 2-2 显示了旅行商问题的计算量随着问题规模 n 的增大而迅速膨胀的情形。

表 2-2 旅行商问题计算量膨胀的数值例

问题规则 n	20	21	22	23	24	25	26	27	28
枚举 $(n-1)!$ 的计算时间	1 秒	20 秒	7 分	2.6 时	2.5 日	2 月	4 年	104 年	2808 年

由于有 n 个城市的旅行商问题共有 $(n-1)!$ 种可行解，假设一台计算机每秒钟可以完成 20 个城市的旅行商问题（即 19! 个）的枚举，则 21 个城市需要 20 秒；22 个城市需要 7 分钟；23 个城市需要 2.6 小时；24 个城市需要 2.5 日；25 个城市需要 2 个月；26 个城市需要 4 年；27 个城市需要 104 年；28 个城市则需要 2808 年；……

因此，由于组合爆炸带来的计算量的膨胀导致大规模现实问题无法找到最优解，只能借助于近似算法求其近似解。

2.2.2 算法与计算量

一个优化问题通常会存在不同的算法，而其中最有效的算法常常用计算所需时间作为衡量标准，这是因为时间决定了算法的效率。算法所用时间取决于该算法的计算量与其输入长度。

计算量是指算法中所包含的加、减、乘、除与比较5种基本运算的次数。算法的输入长度是指计算时二进制输入数据的长度，即优化问题实例所用的计算机内存的单元数。衡量一个算法的好坏通常用计算量与输入长度的大小关系来衡量。

输入长度是按下述方式确定的。对于一个整数 x，其二进制表示是以

$$x = \alpha_s 2^s + \alpha_{s-1} 2^{s-1} + \cdots + \alpha_1 2^1 + \alpha_0 (\alpha_s \neq 0)$$

的系数 $\alpha_s \alpha_{s-1} \cdots \alpha_1 \alpha_0$，$\alpha_i \in \{0,1\}$，$i = 0,1,\cdots,s$ 来表示的。因为，

$$2^s \leqslant 2^{\log_2 x} = x \leqslant 2^{s+1} \leqslant 2^{\log_2 x + 1}$$

即 x 的输入长度为其二进制的位数 $s+1$，在 $\log_2 x$ 与 $\log_2 x + 1$ 之间。如果用 $\lceil x \rceil$ 表示大于或等于 x 的最小整数（向上取整），则 x 的输入长度不超过 $\lceil \log_2 x \rceil + 1$。

以旅行商问题为例，城市数为 n，各城市间的距离为 d_{ij}，$1 \leqslant i,j \leqslant n$，$i \neq j$。$d_{ij}$ 的输入长度不超过 $\lfloor \log_2 d_{ij} \rfloor + 1$。由于这样的数据共有 $n(n-1)$ 个，故旅行商问题的输入总长度不超过 $L = n(n-1)(\lfloor \log_2 d_{ij} \rfloor + 1)$。

通常，优化问题任意实例的输入长度都是其问题规模 n 的多项式函数，由于对输入长度的估计是粗略的以及多项式算法的“封闭性”（参见2.2.3），基本可以用问题的规模 n 来简单代替输入长度 L。以下为简化起见，关于算法计算量的描述都是用问题规模 n 代替其输入长度 L。

计算量包括加、减、乘、除与比较5种基本运算的次数（step数）。由于每种运算所需的基本时间都很短、非常接近，尽管诸如加法等的运算较之乘法运算要花费更多时间，也将其近似为相同。因此，可以用这些基本计算的步数（次数）来统一描述计算量的大小。

如 n 个数的求和运算 $a_1 + a_2 + \cdots + a_n$ 需要做 $n-1$ 次加法，其计算量为 $n-1$ 步；而求和运算 $2a_1 + 2a_2 + \cdots + 2a_n$ 则需要做 n 次乘法和 $n-1$ 次加法，共需 $2n-1$ 步；同样的求和运算 $2(a_1 + a_2 + \cdots + a_n)$ 则仅需要1次乘法和 $n-1$ 次加法，共需 n 步；进一步将 $2a_1 + 2a_2 + \cdots + 2a_n$ 一般化求得 $a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ 也同样需要 $2n-1$ 步；将上式一般化后可以求得 $n \times n$ 阶矩阵相乘的计算量，即积矩阵每一元素的计算量为 $2n-1$ 步，共有 n^2 个这样的元素，共需 $n^2(2n-1)$ 步；求更复杂的集合 $\{a_1, a_2, \cdots, a_n\}$ 的最大部分集合的和的步数为最大 $2^n(n-1)$ 次加法，$2^n - 1$ 次比较，共需 $2^n(n-1) + 2^n - 1$ 步。因此，再复杂的运算，也都可以用这些基本运算的步数来表示。

同理，任何一个优化问题的算法也可以用上述的计算量来表示其计算复杂程度。

以旅行商问题为例。由 n 个城市构成的旅行商问题，任何一个可行解的评价是通过两两城市间的距离的 n 次求和得到的，则枚举所有 $(n-1)!$ 个可行解共需要 $n!$ 步。根据Stirling的近似公式，

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{51840n^3} + O\left(\frac{1}{n^4}\right)\right) \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{2-10}$$

大体上相当于 n^n 次运算。因此，当 n 比较大时，利用枚举法求得最优解几乎是不可能的。只能求助于近似解法。

再以基本的排序问题为例。对由 n 个元素构成整数列集合 $\{a_1, a_2, \cdots, a_n\}$ 进行升（降）序排列。每次经过一次比较运算，对两两元素进行大小对比并调整其升（降）顺序，重复进行这样的比较、调整，形成了二叉树的形状。在二叉树的最底端共有 n 个元素可能的排列数 $n!$ 个分支，而这样一个二叉树的高度 $\log_2(n!)$ 正是 n 个元素重新排列所需要的比较运算的次数。根据