□ 算法理论与应用丛书

近似算法的设计与分析

堵丁柱　葛可一　胡晓东

Design and Analysis of Approximation Algorithms

高等教育出版社
HIGHER EDUCATION PRESS

□ 算法理论与应用丛书

近似算法的设计与分析

堵丁柱　葛可一　胡晓东

JINSI SUANFA DE SHEJI YU FENXI

高等教育出版社·北京
HIGHER EDUCATION PRESS　BEIJING

内容简介

近似算法是处理难解的组合优化问题的一个非常重要和有效的方法。它可以在多项式时间内求得问题的一个解，并使其目标函数值与最优解的目标函数值之比不超过一个常数。本书将通过大量具有代表性的组合优化问题，介绍近似算法设计和分析中的三种主要方法：贪婪算法、限制方法和松弛方法；所讨论的问题来源于不同的研究和应用领域，其中包括通信网络设计、光纤网络、无线自组织网络和传感器网络、生物信息学、社会网络、工业工程和信息管理系统等。此外，本书还将介绍有关组合优化问题不可近似性的一些基本结果。本书的每一章后面都配有相关内容的习题和历史注记。

本书可作为计算机科学和运筹学专业高年级本科生和研究生的近似算法课程的教材，亦可作为相关研究领域科研人员的参考书。

图书在版编目（CIP）数据

近似算法的设计与分析. 堵丁柱，葛可一，胡晓东著. —北京：高等教育出版社，2011.8

ISBN 978-7-04-031967-5

Ⅰ.①近… Ⅱ.①堵… ②葛… ③胡… Ⅲ.①近似计算-研究生-教材 Ⅳ.①O242.2

中国版本图书馆 CIP 数据核字（2011）第 126023 号

策划编辑 赵天夫　　责任编辑 李华英　　封面设计 王凌波　　责任校对 杨雪莲

责任印制 毛斯璐

出版发行	高等教育出版社	咨询电话	400-810-0598
社　　址	北京市西城区德外大街 4 号	网　　址	http://www.hep.edu.cn
邮政编码	100120		http://www.hep.com.cn
印　　刷	北京中科印刷有限公司	网上订购	http://www.landraco.com
开　　本	787mm×1092mm　1/16		http://www.landraco.com.cn
印　　张	27.5	版　　次	2011 年 8 月第 1 版
字　　数	520 000	印　　次	2011 年 8 月第 1 次印刷
购书热线	010-58581118	定　　价	79.00 元

物 料 号　31967-00

前言

求解组合优化问题的近似算法是一种有效算法，它总可以找到问题的一个可行解，其目标函数值与最优解的目标函数值之比不超过某个值。亦即，近似算法并不花费指数时间寻找优化问题的最优解，而是在多项式时间内求得一个近似最优解。虽然人们从 20 世纪 60 年代中期就开始研究这种近似算法，但是直到 **NP**-完全性理论被发现以后，人们才真正理解了其重要性。在 **NP**-完全性理论框架下，人们在一些非常合情理的假设下，证明了许多著名的组合优化问题是非常难处理的，也就是说，在多项式时间内无法找到这些问题的最优解。因此，要处理这些问题，人们所能期望的最好方案就是设计可以找到接近最优解的近似算法。

在过去几十年里，与近似算法相关的研究成果以爆炸般的速度增加。这种增长速度有一部分来源于相关研究领域的快速发展，例如数据挖掘、通信网络、生物信息学和算法博弈理论。这些新发展起来的研究领域催生了一大批新的、难处理的优化问题，其中大部分在实际问题中都有直接的应用，因而人们希望能设计出求解这些优化问题的有效近似算法。

除了来自于外部和实际应用的需要驱动了人们对有效的近似算法的研究，同时近似算法理论内部出现的丰富研究成果及其具有的深刻内涵形成了另外一股内在的驱动力。当人们设计一个可以求出准确解的算法时，评价一个算法性能好坏的主要的标准就是它的运行时间。由于这个固定的标准常常是唯一的，因而就限制了人们设计算法时可以采用的技巧。然而，当人们设计一个近似算法并对其进行分析的时候，还有一个与运行时间同样重要的尺度，即它的性能比。它可以度量用近似算法所能得到的解与最优解的接近程度。有了这个新的尺度，人们设计和分析近似算法时，就增加了一个新的、需要考虑的角度。换言之，人们可以研究如何应用不同的算法设计技巧，在近似算法的运行时间和它的性能比这两个尺度之间实现不同方式的平衡。除此之外，我们还可以探究一个优化问题的近似解的更理论性的一些性质：在采用某种算法设计策略时，一个具体的近似算法的性能比是多少？求解该优化问题的

任意一个多项式时间近似算法的最好性能比又是多少? 是否存在求解该问题的多项式时间近似方案, 甚至完全多项式时间近似方案? 所有这些问题不仅在近似算法的设计和应用中有重要的意义, 而且还具有很大的理论意义, 因为它们与 **NP**-完全性理论有着非常密切的联系。

在研究这些大量的、新发现的优化问题及相应的理论问题的过程中, 人们逐步地发展出设计近似算法的许多新技巧, 包括贪婪策略、限制和划分方法、松弛方法、局部搜索方法、幂图方法、基于线性规划和半定规划的算法及相应的各种舍入技巧。因为我们无法在一本书的篇幅内全面介绍这些方法和相应的结果, 所以在本书中, 我们只针对近似算法设计和分析中的几类主要方法进行研讨。不过, 对每一类方法我们都一一通过大量的具有代表性的优化问题详细地分析研究。实际上, 本书是根据近似算法的不同设计方法, 而不是根据所考虑的优化问题, 来组织各个章节内容的。这样读者就可以对具有相同特质的若干个近似算法一起进行学习, 从而以一种统一的方式了解它们的设计思想。为此, 本书分为五个部分: 首先, 在第一部分, 即**第一章**, 我们简明扼要地介绍 **NP**-完全性和近似算法的概念。在第二部分, 也就是**第二章**, 我们对贪婪算法进行深入的分析, 包括以次模函数为势函数的贪婪算法和以非次模函数为势函数的贪婪算法。第三部分包含三章: **第三章**、**第四章**和**第五章**。在这三章中我们讨论多种限制方法, 其中包含用于处理几何问题的划分和断切方法。第四部分包含**第六章**、**第七章**、**第八章**和**第九章**。在这四章中我们主要讨论松弛方法。在**第六章**中我们对松弛方法进行一般性的讨论以后, 在紧接着的三章中, 讨论基于线性和半定规划的近似算法设计, 包括原始对偶方案和与之等价的局部比值方法。在最后一部分, 即**第十章**, 我们介绍应用 **NP**-完全性理论的近期成果所取得的各种不可近似性结果。

本教科书是根据作者们在中国和美国的几所大学授课的讲义编写而成的, 其中包括美国的明尼苏达大学和得克萨斯大学达拉斯分校, 中国的清华大学、中国科学院研究生院、西安交通大学、浙江大学、华东师范大学、大连理工大学、新疆大学、南开大学、兰州铁道学院、西安电子科技大学和哈尔滨工业大学。本书可用于一年级研究生近似算法课程的教学。在一个学期的课程里, 教师可以讲授前两个部分, 以及第三部分和第四部分中的两章, 再加上**第十章**。对于能力非常强的学生, 授课教师还可以针对第二、三、四部分中的任一部分, 组织相应的组合优化讨论班, 并搜集一些相关方向的最新文献, 作为讨论班的补充材料。例如, 可以结合**第七章**、**第八章**和**第九章**的内容开设一个讨论班, 重点学习和研究基于线性规划和半定规划的近似算法设计。

我们在本书的编写过程中得到来自我们的朋友、同事和学生们的帮助。我们要感谢万鹏俊、伍伟丽、成秀珍、王洁、徐寅峰、张昭、李德英、黄荷娇、朱洪、张国川、王伟、高树刚、高晓枫、邹凤、丁玲、李宪越、麦奕泰 (My T. Thai)、金东炫 (Donghyun Kim)、威尔森 (J. K. Willson) 和苏查耶 (Roozbeh Ebrahimi Soorchaei)。

他们对本书的初稿提出了宝贵的建议和修改意见。我们同时还要感谢储枫 (Francis Yao) 教授、理查德 · 卡普 (Richard Karp) 教授、罗纳德 · 格雷厄姆 (Ronald Graham) 教授和金芳蓉 (Fan Chung) 教授对本书的编写给予的鼓励和支持。我们还要特别感谢姚期智 (Andrew Yao) 教授和清华大学理论计算机科学研究所, 在本书的前两位作者对清华大学的数次访问过程中为他们提供了慷慨的支持和良好的科研环境。最后我们要感谢高等教育出版社的赵天夫先生, 他为本书的撰写和编辑提供了很多的帮助。

堵丁柱 (美国得克萨斯大学达拉斯分校)
葛可一 (美国纽约州立大学石溪分校)
胡晓东 (中国科学院数学与系统科学研究院)
2011 年 3 月

目　录

第一章

引言

受过教育的人的一个标志就是，
他满足于接受事物自身所具有的精确程度，
而当仅能得到近似值时，他不去追求无法获得的精确值。
—— 亚里士多德[1]

当我们很难找到一个优化问题的最优解时，我们就应该考虑如何去找这个问题的一个近似解。在本章中我们要介绍与近似算法相关的一些基本概念。为此我们考虑一个易于表述的优化问题，从而说明如何在近似算法的近似程度和时间效率间进行平衡。我们还会对计算复杂性的一般理论作一个简单的介绍，以说明如何应用计算复杂性的一般理论，并依据优化问题的可近似程度将它们分类。

1.1 “芝麻，开门!”

在《一千零一夜》记述的关于阿里巴巴与四十大盗的传说中，当阿里巴巴念完咒语 “芝麻，开门!” 以后，他发现自己神奇地进入了四十大盗藏宝的洞穴中。狂喜过后，他意识到自己面临着一个优化问题：洞穴里藏的值钱的宝物太多了，但是他随身只有一个背包，他无法将所有的宝物都装进背包一起带走；他应该挑选哪些宝物将它们装进背包带走，而且使得一次能带走的宝物的总价值最大呢?

实际上，依照现代的术语，阿里巴巴遇到的是一个资源管理问题。在这类问题中，我们有 S 个单位的资源 (一个容积为 S 的背包)，和 n 项工作 (拿走洞穴中的 n 件宝

[1] Aristoteles (前 384 — 前 322)：古希腊斯吉塔拉人，哲学家、科学家和教育家。

物)。完成每一项工作都会得到一些收益, 但同时也会消耗掉一定的资源 (每一件宝物都有一定的价值和体积)。我们的目标是, 在消耗的所有资源不超过 S 的条件下, 获得尽可能多的收益。下面我们用数学化的方式描述一下阿里巴巴遇到的问题:

问题 1.1 (阿里巴巴): *任给 n 个物品 $I_1, I_2, \cdots, I_n$, 假定物品 I_i 具有体积 s_i 和价值 c_i, 另给定一个容积为 S 的背包。目标是挑选 n 个物品的一个子集合 $\mathcal{A}$ 使得 $\mathcal{A}$ 中的物品都可以放进背包中, 即 $\sum_{I_i \in \mathcal{A}} s_i \leqslant S$, 且这些物品的价值之和 $\sum_{I_i \in \mathcal{A}} c_i$ 最大。*

如果用一个 (0-1)-变量 x_i 来表示是否将物品 I_i 放进背包中:

$$x_i = \begin{cases} 1, \text{若 } I_i \in \mathcal{A}, \\ 0, \text{若 } I_i \notin \mathcal{A}, \end{cases}$$

那么我们可以将**阿里巴巴问题**表述为如下 (0-1)-规划问题:

问题 1.2 (背包): *任给 $2n+1$ 个正整数 $S, s_1, s_2, \cdots, s_n$ 和 $c_1, c_2, \cdots, c_n$, 求解以下 (0-1)-整数规划:*

$$\begin{aligned} \text{求最大值} \quad & c(\boldsymbol{x}) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n, \\ \text{满足条件} \quad & s_1 x_1 + s_2 x_2 + \cdots + s_n x_n \leqslant S; \\ & x_1, x_2, \cdots, x_n \in \{0, 1\}。 \end{aligned}$$

令 opt 表示目标函数 $c(\boldsymbol{x})$ 的最优值。注意到, 如果 $s_k > S$, 那么一定有 $x_k = 0$, 我们就不必考虑物品 I_k 了。因此, 不失一般性, 我们可以假定对于所有 $k = 1, 2, \cdots, n$, 有 $s_k \leqslant S$。由这个假定可知, 对于所有 $k = 1, 2, \cdots, n$, 有 $opt \geqslant c_k$。

处理**背包问题**有许多种不同类型的算法。我们下面先介绍一个基于动态规划的算法, 它可以求出**背包问题**的最优解。

为了叙述方便, 我们先定义几个记号。对于子集合 $I \subseteq \{1, 2, \cdots, n\}$, 令 S_I 表示编号在 I 中的所有物品的体积之和 $S_I = \sum_{k \in I} s_k$。对 $\{i, j\}, 1 \leqslant i \leqslant n$ 且 $0 \leqslant j \leqslant \sum_{i=1}^{n} c_i$, 如果存在子集合 $I \subseteq \{1, 2, \cdots, i\}$ 使得

$$\sum_{k \in I} c_k = j \text{ 且 } S_I \leqslant S,$$

那么令 $c(i, j)$ 为使得 S_I 达到最小的指标集合 I; 否则认为 $c(i, j)$ 无定义, 记为 $c(i, j) = \text{nil}$。

根据上述定义, 显然有

$$opt = \max\{j \mid \mathrm{c}(n, j) \neq \text{nil}\}。$$

因此, 要想找到**背包问题**的一个最优解, 我们只需计算出所有的 $c(i, j)$。下面的算法就是根据这个思想设计的[①]。

[①]我们在本书中总是用标准的伪码描述算法, 具体用法说明参见考曼等人 [75] 所编写的教科书。

算法 1.A (求解背包问题的精确算法)

输入: 正整数 $S, s_1, c_1, s_2, c_2, \cdots, s_n, c_n$。

(1) 置 $c_{\text{sum}} \leftarrow \sum_{i=1}^{n} c_i$。

(2) **对** $j = 0, 1, \cdots, c_{\text{sum}}$ **执行**

若 $j = 0$ **则置** $c(1, j) \leftarrow \varnothing$

否则 若 $j = c_1$ **则置** $c(1, j) \leftarrow \{1\}$

否则置 $c(1, j) \leftarrow \text{nil}$。

执行—结束

(3) **对** $i = 2, 3, \cdots, n$ **执行**

对 $j = 0, 1, \cdots, c_{\text{sum}}$ **执行**

若 $[c(i-1, j-c_i) \neq \text{nil}]$ **且** $[S_{c(i-1,j-c_i)} \leqslant S - s_i]$

且 $[c(i-1, j) \neq \text{nil} \Rightarrow S_{c(i-1,j)} > S_{c(i-1,j-c_i)} + s_i]$

则置 $c(i, j) \leftarrow c(i-1, j-c_i) \cup \{i\}$

否则置 $c(i, j) \leftarrow c(i-1, j)$。

执行—结束

执行—结束

(4) 输出: $c^* \leftarrow \max\{j | c(n, j) \neq \text{nil}\}$。

不难验证, 对于任意输入的**背包问题**的一个实例, 上述算法总能求出它的一个最优解 (**习题 1.1**)。

下面我们考虑**算法 1.A** 的时间复杂度。因为阿里巴巴必须在四十大盗回来之前, 装满宝物并离开洞穴, 所以他需要一个有效的算法。很容易看出, 对于任意一个子集合 $I \subseteq \{1, 2, \cdots, n\}$, 计算 S_I 需要时间 $O(n \log S)$ [①]。因此, **算法 1.A** 求出**背包问题**的最优解需要时间 $O(n^3 M \log(MS))$, 这里 $M = \max\{c_k | 1 \leqslant k \leqslant n\}$。又注意到, 若所有输入的正整数用二进制表示, 则**背包问题**的输入需要占用空间 $n \log M + \log S$。因此, **算法 1.A** 不是一个*多项式时间算法*。它实际上是一个*伪多项式时间算法*, 它的运行时间是问题的最大输入值的一个多项式 (而不一定是整个问题的输入的占用空间的一个多项式)。因为问题的最大输入值可以是一个非常大的数, 所以伪多项式时间算法并不是一个 (时间) 有效的算法。事实上, 如果阿里巴巴用这个算法来挑选他要带走的宝物, 那么即使是用最现代的计算机, 他也无法在四十大盗回来以前, 算出最优的选择。

作为解决这个难题的一个折中方案, 阿里巴巴会觉得一个可以快速算出近似最优选择 (而不是非要得到一个最优选择) 的近似算法对他更有用处。下面的这个算法正好是他需要的。它用了一个简单的贪婪的策略选择带走的宝物, 即优先考虑单位体积具有最大价值的宝物。

[①] 在本书中我们总是用 $\log k$ 表示 $\log_2 k$。

算法 1.B (求解**背包问题**的贪婪算法)

输入: 如**算法 1.A**。

(1) 依据 c_i/s_i 的大小按照递减序排列所有物品。不妨设 $c_1/s_1 \geqslant c_2/s_2 \geqslant \cdots \geqslant c_n/s_n$。

(2) **若** $\sum_{i=1}^{n} s_i \leqslant S$, **则输出**: $c_G \leftarrow \sum_{i=1}^{n} c_i$

否则求出最大指标 k 满足 $\sum_{i=1}^{k} s_i \leqslant S < \sum_{i=1}^{k+1} s_i$

输出: $c_G \leftarrow \max\{c_{k+1}, \sum_{i=1}^{k} c_i\}$。

易知, 上述贪婪算法可以在 $O(n\log(nMS))$ 时间内求出一个解, 因此从它的运行时间上看, 它的效率非常高; 而且下面的定理说明, 它所得到的近似解的目标函数值与最优解的目标函数值相差不是很大。

定理 1.1 *任给***背包问题***的一个实例, 设 opt 是其最优解的目标函数值, c_G 是由***算法 1.B*** 生成的近似解的目标函数值, 则 $opt \leqslant 2c_G$* (我们称**算法 1.B** 的性能比不超过常数 2)。

证明. 若 $\sum_{i=1}^{n} s_i \leqslant S$, 则 $c_G = opt$。因此, 我们不妨假设 $\sum_{i=1}^{n} s_i > S$。设 k 是**算法 1.B** 在第 (2) 步中得到的整数。我们要证明下面两个不等式

$$\sum_{i=1}^{k} c_i \leqslant opt < \sum_{i=1}^{k+1} c_i。 \tag{1.1}$$

第一个不等式显然成立。至于第二个不等式, 注意到在第 (1) 步中, 算法是依据物品价值与体积的比值 c_i/s_i, 按照该值递减的序将物品排列。因此, 假如我们将每个物品分成若干个小块, 那么利用背包的最有效的方法就是, 先装前 k 个物品, 然后再装第 $k+1$ 个物品的若干个小块, 直到将背包装满; 这是因为若用其他物品替换这些小块, 只能降低整个背包的物品价值与体积的比值。这意味着, 所装物品的价值总和少于 $\sum_{i=1}^{k+1} c_i$, 因此它是最优值 opt 的一个上界。

我们同样可以用线性规划的语言来叙述这个证明。事实上, 如果我们用不等式约束 $0 < x_i < 1$ 替换等式约束 $x_i = 0$ 或者 1, 那么我们会得到线性规划的最大值 $\hat{c} \geqslant opt$。很容易验证用下述方法给变量 $\boldsymbol{x}$ 赋值, 可以得到线性规划的一个最优解[①]:

$$x_j = \begin{cases} 1, & j = 1, 2, \cdots, k, \\ \dfrac{S - \sum_{i=1}^{k} s_i}{s_{k+1}}, & j = k+1, \\ 0, & j = k+2, k+3, \cdots, n。 \end{cases}$$

因此可得

$$opt \leqslant \hat{c} = \sum_{i=1}^{k} c_i + \frac{c_{k+1}}{s_{k+1}}\left(S - \sum_{i=1}^{k} s_i\right) < \sum_{i=1}^{k} c_i + \frac{c_{k+1}}{s_{k+1}} s_{k+1} = \sum_{i=1}^{k+1} c_i。$$

[①]在**第七章**我们会详细讲述如何求解线性规划。

最后，由不等式 (1.1) 可以得到

$$c_G = \max\left\{c_{k+1}, \sum_{i=1}^{k} c_i\right\} \geqslant \frac{1}{2}\sum_{i=1}^{k+1} c_i > \frac{opt}{2}。$$

定理证毕。 □

从对以上两个算法的分析中可以看出，在设计算法时，我们需要在算法的运行时间和所得解的精度之间权衡：如果我们对解的精确性降低一点要求，那么我们有可能设计一个非常有效的算法。为了进一步说明权衡近似算法的运行时间和近似程度的思想，我们下面给出求解**背包问题**的其他几个算法，它们的运行时间不同，得到的解与最优解的近似程度也不同。

下面我们说明如何将上述的贪婪算法进行推广，使其可以得到一个更好的近似解，尽管它的时间复杂度变差了，但它仍然还是多项式时间算法。基本思想是这样的：将所有的物品分成两组，价值 c_i 不超过 a 的物品分在一组，价值 c_i 超过 a 的物品分在另一组，这里 a 是一个预先设定的常数。注意到，对任意一个可行解，价值超过 a 的物品最多有 $opt/a \leqslant 2c_G/a$ 个。因此，我们可以在第二组中考虑物品数不超过 $2c_G/a$ 的所有子集合 $I \subseteq \{1,2,\cdots,n\}$；对每个这样的 I，于剩余的空间用贪婪算法在第一组中求一个近似解。这样我们得到了不超过 $n^{2c_G/a}$ 个近似解。在其中找个最大的。这个算法的运行时间仍然是问题输入 (占有空间大小) 的一个多项式。在下面的算法描述中，$|A|$ 表示有限集合 A 中所含有的元素的个数。

算法 1.C (求解**背包问题**的推广的贪婪算法)

输入：如**算法 1.A**；及一个常数 $0 < \varepsilon < 1$。

(1) 运行**算法 1.B** 得到一个近似解 c_G。

(2) 置 $a \leftarrow \varepsilon \cdot c_G$。

(3) 置 $I_a \leftarrow \{i \mid 1 \leqslant i \leqslant n, c_i \leqslant a\}$；重新标号使得 $I_a = \{1,2,\cdots,m\}$ $(m \leqslant n)$。

(4) 将 I_a 中的物品依照 c_i/s_i 的递减序排列；

再重新标号使得 $c_1/s_1 \geqslant c_2/s_2 \geqslant \cdots \geqslant c_m/s_m$。

(5) **对** 每一个满足 $|I| \leqslant 2/\varepsilon$ 集合 $I \subseteq \{m+1, m+2, \cdots, n\}$ **执行**

若 $\sum_{i\in I} s_i > S$ **则** 置 $c(I) \leftarrow 0$

否则若 $\sum_{i=1}^{m} s_i \leqslant S - \sum_{i\in I} s_i$

则 置 $c(I) \leftarrow \sum_{i=1}^{m} c_i + \sum_{i\in I} c_i$

否则 求最大的 k 使得 $\sum_{i=1}^{k} s_i \leqslant S - \sum_{i\in I} s_i < \sum_{i=1}^{k+1} s_i$；

置 $c(I) \leftarrow \sum_{i=1}^{k} c_i + \sum_{i\in I} c_i$

执行—结束

(6) **输出**：$c_{GG} \leftarrow \max\{c(I) | I \subseteq \{m+1, m+2, \cdots, n\}, |I| \leqslant 2/\varepsilon\}$。

定理 1.2 设背包问题的最优值是 opt。则**算法 1.C** 可在 $O(n^{1+2/\varepsilon})$ 时间内求得一个近似解, 其目标函数值 c_{GG} 满足 $opt \leqslant (1+\varepsilon)c_{GG}$。

证明. 设最优解中所有物品的指标集合是 I^*, 即有

$$\sum_{i\in I^*} c_i = opt \quad 和 \quad \sum_{i\in I^*} s_i \leqslant S。$$

令 $\bar{I} = \{i \in I^* | c_i > a\}$, 则有

$$|\bar{I}| \leqslant \frac{opt}{a} \leqslant 2\frac{c_G}{a} = \frac{2}{\varepsilon}。$$

因此在**算法 1.C** 的第 (5) 步中, 指标集 I 最后一定会取为 $\bar{I}$; 再由**定理 1.1** 的证明可得不等式

$$c(\bar{I}) \leqslant opt \leqslant c(\bar{I}) + a。$$

又因为 c_{GG} 是所有解 $c(I)$ 中价值最大的, 所以有

$$c(\bar{I}) \leqslant c_{GG} \leqslant opt \leqslant c(\bar{I}) + a \leqslant c_{GG} + a。$$

假定 C_G 是**算法 1.B** 所得物品指标集合。令 $I_G = \{i \in C_G | c_i > a\}$, 则 $c(I_G) = c_G$。同时, $|I_G| \leqslant c_G/a \leqslant 1/\varepsilon$。因此有 $c_G \leqslant c_{GG}$。由此可得

$$opt \leqslant c_{GG} + a = c_{GG} + \varepsilon \cdot c_G \leqslant (1+\varepsilon)c_{GG}。$$

另外, 因为最多只有 $n^{2/\varepsilon}$ 个指标集合 I 所含的物品个数 $|I| \leqslant 2/\varepsilon$, 所以**算法 1.C** 的运行时间是 $O(n^{1+2/\varepsilon}\log(nMS))$。定理证毕。 □

由**定理 1.2** 可知, 对任意实数 $\varepsilon > 0$, **算法 1.C** 的运行时间为 $O(n^{1+2/\varepsilon}\log(nMS))$, 这是关于 n 的一个多项式。然而, 当 ε 趋于零时, 它的运行时间是 $1/\varepsilon$ 的一个指数函数。我们能不能将关于 $1/\varepsilon$ 增长的算法运行时间降下来呢? 答案是肯定的。我们在下面就给出这样一个近似算法。

算法 1.D (求解背包问题的权衡算法)

输入: 如**算法 1.A**; 正整数 $h > 0$。

(1) 置 $M \leftarrow \max\{c_i | 1 \leqslant i \leqslant n\}$;

对 $k = 1, 2, \cdots, n$ **执行** 置 $c'_k \leftarrow \lfloor c_k n(h+1)/M \rfloor$。

(2) 运行**算法 1.A** 求解下面的**背包问题**:

$$\begin{aligned} &求最大值 \quad c'_1x_1 + c'_2x_2 + \cdots + c'_nx_n, \\ &满足条件 \quad s_1x_1 + s_2x_2 + \cdots + s_nx_n \leqslant S, \\ &\qquad\quad x_1, x_2, \cdots, x_n \in \{0,1\}。 \end{aligned}$$

得到其最优解 $(x_1^*, x_2^*, \cdots, x_n^*)$ (即它的指标集合与 $(c')^*$ 对应)。

(3) 输出: $c_{PT} \leftarrow c_1x_1^* + c_2x_2^* + \cdots + c_nx_n^*$。

定理 1.3 *设背包问题的最优值是 opt, 则由算法 1.D 求得的近似解的目标函数值 c_{PT} 满足 $opt/c_{PT} \leqslant 1 + 1/h$。*

证明. 如**定理 1.2** 的证明, 设最优解 Opt 中所有物品的指标集合是 I^*, 即有

$$\sum_{i\in I^*} c_i = c_{opt} \quad 和 \quad \sum_{i\in I^*} s_i \leqslant S。$$

另设在**算法 1.D** (2) 中求得的指标集合为 J^*, 即 $J^* = \{k|x_k^* = 1, 1 \leqslant k \leqslant n\}$, 则有

$$\begin{aligned}
c_{PT} = \sum_{k\in J^*} c_k &= \sum_{k\in J^*} \frac{c_k n(h+1)}{M} \cdot \frac{M}{n(h+1)} \\
&\geqslant \sum_{k\in J^*} \left\lfloor \frac{c_k n(h+1)}{M} \right\rfloor \cdot \frac{M}{n(h+1)} \\
&= \frac{M}{n(h+1)} \sum_{k\in J^*} c'_k \\
&\geqslant \frac{M}{n(h+1)} \sum_{k\in I^*} c'_k \\
&\geqslant \frac{M}{n(h+1)} \sum_{k\in I^*} \left(\frac{c_k n(h+1)}{M} - 1 \right) \\
&\geqslant opt - \frac{M}{h+1} \\
&\geqslant opt\left(1 - \frac{1}{h+1}\right)。
\end{aligned}$$

上面的第二个不等式成立是因为 J^* 是改变参数得到的**背包问题**的最优解的指标集; 而最后一个不等式成立是因为 $M = \max_{1\leqslant i\leqslant n}\{c_i\} \leqslant opt$。将以上不等式简单整理一下, 即可证明定理。 □

注意在**算法 1.D** (2) 中, 求解改变参数得到的**背包问题**需要时间 $O(n^3 M' \log(M'S))$, 这里 $M' = \max\{c'_k \mid 1 \leqslant k \leqslant n\} \leqslant n(h+1)$。因此该算法的运行时间是 $O(n^4 h \log(nhS))$, 这是一个关于 n, $\log S$, 和 $h = 1/\varepsilon$ 的多项式。因此, 就**算法 1.D** 的运行时间和近似程度而言, 它比推广的贪婪算法 (即**算法 1.C**) 要好。

由上面的讨论, 我们可以看到, 如果我们将目光从最优解转移到近似解, 那么我们会发现许多新的思想和方法可用于求解优化问题。事实上, 设计和分析近似算法与设计和分析精确 (或者最优) 算法[①]有很大不同; 这是一个藏有丰富新奇矿藏的洞穴。让我们一起念 “芝麻, 开门”, 然后走进洞穴, 一探究竟吧。

[①]本书中指可以求得最优解的算法。