2022 年 3 月　　运 筹 学 学 报　　第 26 卷 第 1 期
Mar., 2022　　Operations Research Transactions　　Vol.26 No.1

DOI: 10.15960/j.cnki.issn.1007-6093.2022.01.005

经典一维装箱问题近似算法的研究进展*

陈　婳[1,†]　张国川[2]

摘要　自20世纪70年代开始, 随着计算复杂性理论的建立, 近似算法逐渐成为组合优化的重要研究方向。作为第一批研究对象, 装箱问题引起了组合优化领域学者的极大关注。装箱问题模型简单、拓展性强, 广泛出现在各种带容量约束的资源分配问题中。除了在物流装载和材料切割等方面愈来愈重要的应用外, 装箱算法的任何理论突破都关乎到整个组合优化领域的发展。直到今天, 对装箱问题近似算法的研究仍如火如荼。本文主要针对一维模型, 简述若干经典 Fit 算法的发展历程, 分析基于线性规划松弛的近似方案的主要思路, 总结当前的研究现状并对未来的研究提供一些参考建议。

关键词　装箱问题, 近似算法, 线性规划松弛

中图分类号　O221.7

2010 数学分类号　90C27, 90C59

A survey on approximation algorithms for one dimensional bin packing*

CHEN Hua[1,†]　ZHANG Guochuan[2]

Abstract　With the emergence of computational complexity theory, the study of approximation algorithms has gradually become an important field in combinatorial optimization since the 1970s. As one of the classic combinatorial optimization problems, bin packing has attracted great attention. It can be widely found in various resource allocation problems with capacity constraints. Its more and more important applications including logistics loading and material cutting aside, any theoretical breakthrough of bin packing algorithms is related to the development of the whole combinatorial optimization. At present, the research on approximation algorithms of bin packing is still popular. This paper briefly introduces the process of some classic Fit algorithms, analyzes the main ideas of approximation schemes based on linear programming relaxation, reviews the state of the art, and provides some suggestions for future research.

Keywords　bin packing, approximation algorithms, linear programming relaxation

Chinese Library Classification　O221.7

2010 Mathematics Subject Classification　90C27, 90C59

20世纪80年代末越民义先生关注到装箱问题的近似算法, 开始研究最为经典的 FFD

收稿日期：2021-06-07

* 基金项目: 国家自然科学基金 (Nos. 11771365)

1. 浙江大学数学科学学院, 浙江杭州 310058; School of Mathematical Sciences, Zhejiang University, Hangzhou 310058, Zhejiang, China

2. 浙江大学计算机科学与技术学院, 浙江杭州 310058; College of Computer Sciences and Technology, Zhejiang University, Hangzhou 310058, Zhejiang, China

† 通信作者　E-mail: chenhua_by@zju.edu.cn

算法。1991 年发表了“A simple proof of the inequality $\mathrm{FFD}(L) \leqslant \frac{11}{9}\mathrm{OPT}(L)+1, \forall L$ for the FFD bin-packing algorithm” 的文章[1], 通过权函数的改良和精准分析, 将 FFD 算法近似界的尾项从 3 降到 1, 并大大缩减了证明的篇幅。自此国内关于组合优化问题近似算法的研究开始兴起。谨以此文介绍经典一维装箱问题的勃勃生机, 并向越民义先生致敬!

1 概述

装箱是一类与任务安排和资源配置相关的组合优化问题, 主要包含两大要素: 物品 (任务) 和箱子 (资源)。经典的装箱问题具体描述为: 给定若干个带有尺寸的物品, 要求将所有物品放入容量给定的箱子中, 使得每个箱子中的物品尺寸之和不超过箱子容量并使所用的箱子数目最少。简单起见, 将上述模型标准化: 给定若干个尺寸在 0 到 1 之间的物品, 目标是使用数量尽可能少的单位容量箱子装下所有物品。

装箱问题广泛出现在许多真实的应用场景中。从带重量限制的货车装载问题到物料裁剪等工业生产中的问题[2], 再到计算机科学中的内存分页管理、文件分配问题等都可归为装箱问题。众所周知, 装箱问题是 NP 困难的, 即在 $\mathrm{P} \neq \mathrm{NP}$ 的假设下, 多项式时间内无法找到该问题的最优解或者说该问题不存在多项式时间算法。此时, 考虑牺牲问题解的最优性继而寻找与最优解相差不大且可在多项式时间内得到的近似解是不错的选择。这种求解问题近似解的方法称为近似算法。不同的近似算法获得的近似解的性能是有差异的, 一般我们用 “近似比” 的概念来衡量算法的优劣。对装箱之类的极小化问题, 算法的近似比定义为最坏情况下算法的求解值与问题的最优值之间的比率。对任意实例 I, 用 $A(I)$ 表示针对实例 I 运行算法 A 后所用箱子数; $\mathrm{OPT}(I)$ 表示装完实例 I 中所有物品所用的最少的箱子数, 即实例 I 上的最优值。也就是说, 若对任意实例 I, $A(I) \leqslant \alpha \cdot \mathrm{OPT}(I)$, 则称算法 A 的近似比至多是 α。近似比也常称为绝对近似比。

在装箱问题的研究中, 人们更看重的是在最优值较大的情况下, 算法求解值和最优值之间的比值。此时大家关注的不再是所有的实例, 而是实例中最优值较大的那些, 该比值就是渐近近似比。若对任意实例 I, $A(I) \leqslant \alpha \cdot \mathrm{OPT}(I)+k$, 其中 $k=o(\mathrm{OPT}(I))$, 则称算法 A 的渐近近似比至多是 α。满足上式的所有 α 的下确界即是算法 A 的渐近近似比, 记为 r_A^{∞}。

装箱问题中若所有的物品信息已知, 则它是离线问题; 若物品逐个到达, 即时安排, 而我们对未到达物品信息一无所知, 且做出的决定无法更改, 此时称为在线问题。上面定义的两种近似比通常用来描述离线问题近似算法的性能。而对于在线问题, 一般用 “竞争比” 的概念。竞争比定义为最坏情形下某在线算法的求解值与问题的最优值之间的比率。同样, 算法渐近竞争比也是针对在线问题而言的。

以上所述为经典的一维装箱问题。自然地, 可将问题推广到多维情况。常见的有两种: 几何装箱和向量装箱。在几何装箱中, 箱子是 n 维的正方体 (边长为 1, $n \leqslant 3$), 物品(长方体)在 n 个维度上都有尺寸 (不超过 1), 要求在不能重叠的条件下装完所有物品使用的箱子数量尽可能少; 在向量装箱中, 箱子的容量为各个维度上都是 1 的 n 维向量, 物品是每个维度上尺寸在 0 到 1 之间的 n 维向量, 要求装在同一个箱子里的物品同一个维度上的尺寸之和不超过 1, 装完所有物品使所用的箱子个数尽可能少。

在 $\mathrm{P} \neq \mathrm{NP}$ 的假设下, NP 困难问题不存在多项式时间的最优算法(绝对近似比为 1)。于是, $(1+\varepsilon)$ 是在理论上能做到的最好可能的绝对近似比, 其中 $\varepsilon > 0$ 任意小。具有

这样近似结果的一簇近似算法为 PTAS (多项式时间近似方案)。同样, 若存在一簇算法 $\{A_\varepsilon : \varepsilon > 0\}$ 具有 $(1+\varepsilon)$ 的渐近近似比, 称这簇算法为 APTAS (渐近多项式时间近似方案)。上述两者运行时间的要求均为 $O(n^{f(\frac{1}{\varepsilon})})$, 其中 $f(\frac{1}{\varepsilon})$ 为关于 $\frac{1}{\varepsilon}$ 的函数, 即当 $\varepsilon > 0$ 固定后, 运行时间为多项式。

装箱问题的研究始于 20 世纪 70 年代。Ullman[3] 是最早开始研究装箱问题近似算法的; Johnson 在其博士论文[4] 中所用的权函数方法开创了后续许多理论研究的先河; 在 1981 年, Garey 和 Johnson[5] 曾对装箱问题做过综述, 包括一维情形下的性能分析及其许多变形和推广; 三年后, Coffman 等人[6] 在此基础上添加和更新了一些新算法和新结果; 同一批作者在 1996 年又对装箱问题做了一次综述[7], 虽然主要工作还是在经典的一维问题上, 但较前两次综述来说增添了在线算法以及其他新算法的结果, 内容极大丰富了。Coffman 等人[8] 在 2013 年对于经典一维装箱问题做了比较全面的介绍。内容涵盖在线算法、离线算法、变尺寸装箱、自私装箱博弈以及装箱问题的对偶问题等方面。可以看到, 每次综述都有更多有趣的问题模型和算法涌现出来。

装箱问题的研究无论是在计算机科学还是运筹学中都具有非常重要的意义。此问题的研究见证了近似算法和在线算法的发展历程, 已成为组合优化问题的基础和经典。近年来, 装箱问题近似算法的研究一直在逐步完善。一维装箱问题的研究是各类纷繁复杂的装箱问题的研究基础, 对其现阶段的一些重要研究进行综述很有必要。本文将主要着眼于一维离线装箱问题中出现的经典 Fit 算法和线性规划舍入算法两大类展开讨论, 希望能带领大家追寻经典算法的发展脉络, 感受它的独特魅力。

2　Fit 算法

就装箱问题而言, 最直接的方法莫过于按照某种简单规则依次将物品放入有足够剩余空间的箱子中 (必要时打开新的箱子)。这样的算法统称为 Fit 算法。考虑一个最优的装箱方案, 将使用的箱子依次编号, 同时将物品进行如下编号: 箱子编号小的物品在前; 同一个箱子的物品可任意编号。不妨设相邻两个箱子中后面箱子编号最小的物品无法装入前面的箱子中 (否则可移动该物品, 得到的仍是最优装箱)。容易看出, 按照物品编号从小到大排列好, 再使用下面的算法可以得到完全相同的最优解:

Next Fit (NF) 算法 打开一个新的箱子, 将第一个物品装进箱子。当后续物品到达时, 如果当前打开的箱子有足够的空间, 就直接放入; 否则关闭该箱子 (不再接受任何新的物品), 打开一个新的箱子, 将该物品放进去。

NF 算法任何时候仅保持一个打开的箱子。不难看出 NF 是线性时间算法。如果物品安排的顺序是任意的, 容易证明此算法的绝对近似比和渐近近似比均为 2。上界来自于下面的事实: 相邻两个箱子的物品尺寸总和大于 1。下界源自下面的实例[4]: $\frac{1}{2}, \varepsilon, \frac{1}{2}, \varepsilon, \frac{1}{2}, \varepsilon, \cdots$; 共有 m 组 $\frac{1}{2}, \varepsilon$, 其中 $\varepsilon > 0$ 充分小, m 充分大, 箱子容量为 1。算法解为 m, 最优解为 $\frac{m}{2} + 1$。稍微细致一点的分析可以得出, 对任意实例 I, $\mathrm{NF}(I) \leqslant 2\mathrm{OPT}(I) - 1$。

虽然 NF 算法简易快速, 但其近似效果较差。容易看出 NF 算法仅保持一个打开的箱子, 提前关闭的箱子的剩余空间没有得到利用。如果我们在装箱结束之前不关闭任何箱子, 是否可以做得更好呢? 下面我们将介绍几个算法, 它们均利用到前面已使用箱子的空间。

Any Fit (AF) 算法 算法称为 Any Fit, 若其满足如下性质: 当物品到达时, 所有目

前已经打开的箱子均无法装下该物品的情况下, 才允许新开一个箱子。

根据上面的性质, 对当前要考虑的物品而言, 如果已打开的箱子空间都不够, 那么就需要一个新箱子。否则我们必须在有足够空间的已打开箱子中选择一个。依据不同的选择规则就得到不同的算法。

- First Fit (FF) 算法: 选择最早打开的箱子(编号最小)。
- Best Fit (BF) 算法: 选择最满的箱子(剩余空间最小)。
- Worst Fit (WF) 算法: 选择最空的箱子(剩余空间最大)。

上述三种 Any Fit 算法都满足相邻两个箱子的物品尺寸总和大于 1, 故它们近似比都不会比 NF 差。但容易发现, 前面 NF 算法的实例也适用于 WF 算法, 从而证明 WF 和 NF 一样差。Johnson[9] 证明了 $r_{\mathrm{FF}}^{\infty} \leqslant r_{\mathrm{AF}}^{\infty} \leqslant r_{\mathrm{WF}}^{\infty}$, 其中$r_{A}^{\infty}$ 表示算法 A 的渐近近似比。

与 WF 不同, FF 和 BF 算法满足一个更强的性质: 设 k 是当前打开箱子中剩余空间最大的箱子的最大编号。也就是说, 第 k 个打开的箱子剩余空间最大, 和它有相同剩余空间的箱子打开得比它早(编号小于 k), 那么除非当前物品无法装进前 $k-1$ 个箱子里, 否则它不会装进第 k 个箱子里。满足此性质的算法称为 Almost Any Fit (AAF) 算法。FF 和 BF 是 AAF 算法, 而 NF 和 WF 不是。Johnson[9] 证明了 $r_{\mathrm{FF}}^{\infty}=r_{\mathrm{AAF}}^{\infty}$。

我们很容易修正 WF 将其变成 AAF 算法。修正后的算法记为 Almost Worst Fit (AWF): 将当前物品放入能装下它的剩余空间第二大的箱子中; 若这样的箱子不存在, 便将其放入能装它的剩余空间最大的箱子中; 若上述两种情况都不存在, 则新打开一个箱子将其装入。AWF 算法看起来只是对 WF 算法的微小修正, 但可观察到 AWF 属于 AAF 算法, 故 $r_{\mathrm{FF}}^{\infty}=r_{\mathrm{AWF}}^{\infty}$。

从定义来看 BF 算法似乎比 FF 算法有更高的箱子利用率, 但并非总是如此。考虑下面 5 个物品的实例: 0.5, 0.7, 0.1, 0.4, 0.3。此实例 FF 算法需要 2 个箱子, 而 BF 算法则需要 3 个。下面我们来回顾经典 AAF 算法的研究历程。

Ullman[3] 是最早开始研究 FF 算法和 BF 算法的。在 1971 年, 他证明了对于装箱问题的任意实例 I, $\mathrm{FF}(I) \leqslant 1.7\mathrm{OPT}(I)+3$ 且 $\mathrm{BF}(I) \leqslant 1.7\mathrm{OPT}(I)+3$. Johnson 等人[10] 在 1974 年给出了 FF 算法渐近近似比恰为 1.7 的实例, 且证明了对于任意实例 I, $\mathrm{FF}(I) \leqslant 1.7\mathrm{OPT}(I)+2$。Garey 等人[11] 在 1976 年证明了对任意实例 I, $\mathrm{BF}(I) \leqslant \lceil 1.7\mathrm{OPT}(I) \rceil$。

Dósa 和 Sgall[12] 在 2013 年用精致的权函数方法证明了 FF 算法的绝对近似比为 1.7, 即对任意实例 I, $\mathrm{FF}(I) \leqslant \lfloor 1.7\mathrm{OPT}(I) \rfloor$。第二年, Dósa 和 Sgall[13] 证明了 BF 算法的绝对近似比也为 1.7。这两篇文章把权函数方法发挥得淋漓尽致, 证明了 FF 和 BF 算法的渐近近似比和绝对近似比相等, 多少有点令人意外。

带空间限制的 Fit 算法 前面提到, NF 算法只保持一个最近打开的箱子, 而 AF 算法在运行过程中则保持所有已打开的箱子。如果我们限定仅允许一定数量的箱子处在打开状态, 也就是说, 当打开的箱子超过一定量, 必须关闭一些, 会有什么影响呢? 容易看出, 如果我们只允许保持一个箱子打开, 那么 NF 就是最好的算法。如果我们允许保持 $k \geqslant 2$ 个箱子, 而当打开的 k 个箱子均不能装下当前物品时, 我们必须关闭一个箱子才能打开新的箱子。可以预见, 不同的关闭原则也将产生不同的算法。考虑两个常见的原则: 总是关闭最先打开的箱子 (First Close), 总是关闭剩余空间最小的箱子 (Best Close)。这两个关闭原则刚好与装箱原则 First Fit 和 Best Fit 对应。它们两两组合构成了四个算法, 统一记成 AXY_k 算法, 其中 X 代表装箱规则, Y 代表关闭规则, k 则为可以保持打开的箱子个数。X={B,F| B=Best Fit, F=First Fit}, Y={B,F| B=Best Close, F=First Close}。这样,

四种算法分别是: AFF_k, ABF_k, AFB_k, ABB_k。

AFF_k 算法, 也称为 Next-k-Fit (NF_k) 算法, 是 NF 算法的一个推广。该算法由 Johnson[4] 提出。他证明了 $r^{\infty}_{\mathrm{NF}_k} \in \left[1.7+\frac{3}{10k}, 2\right]$。Csirik 和 Imreh[14] 进一步分析出 $r^{\infty}_{\mathrm{NF}_2}=2$; $k\geqslant 3$ 时, $r^{\infty}_{\mathrm{NF}_k} \in \left[1.7+\frac{3}{10(k-1)}, 1.75+\frac{7}{4(2k+3)}\right]$。最后 Mao[15] 给出了紧界, 即对任意 $k\geqslant 2$ 均有 $r^{\infty}_{\mathrm{NF}_k}=1.7+\frac{3}{10(k-1)}$。$\mathrm{ABF}_k$ 算法是由 Mao[16] 首先分析的, 她证明了对任意 $k\geqslant 2$ 均满足 $r^{\infty}_{\mathrm{ABF}_k}=1.7+\frac{3}{10k}$。Zhang 和 Yue[17] 则证明了对任意 $k\geqslant 2$, $r^{\infty}_{\mathrm{AFB}_k}=r^{\infty}_{\mathrm{NF}_k}=1.7+\frac{3}{10(k-1)}$。也就是说, 在 First Fit 的装箱规则下, Best Close 并没有什么帮助。与之对比的是, 当 $k\geqslant 2$, Csirik 和 Johnson[18] 证明了算法 ABB_k 与 FF 和 BF 有相等的渐近近似比 1.7。也就是说 Best Close 与 Best Fit 结合, 只需保持两个打开的箱子即可以达到与 FF 和 BF 相同的渐近近似性能。

NF 算法的另一个重要推广为 Harmonic-k-Fit (HF_k) 算法, 由 Lee 和 Lee[19] 于 1985 年提出的: 将 $(0,1]$ 区间分成 $I_j=(\frac{1}{j+1},\frac{1}{j}], j=1,\cdots,k-1$ 和 $I_k=(0,\frac{1}{k}]$ 共 k 个区间。若物品的尺寸大小在某 I_j 中, 则称此物品为 I_j-元。同样箱子也被分成了 k 种类型, 第 j 种类型的箱子 I_j-箱只能为 I_j-元使用, 每个 I_j-元均按照 NF 规则装进 I_j-箱。在任何时刻, 至多有 k 个箱子是保持打开的。当 $k=1$ 时, 它就是 NF 算法; $k=2$ 时, 利用 NF 渐近近似比同样的方法, 容易证明 $r^{\infty}_{\mathrm{HF}_2}=2$。Lee 和 Lee[19] 证明了当 $k>6$, 算法 HF_k 的渐近近似比严格小于 1.7; 当 $k\to\infty$ 时, $r^{\infty}_{\mathrm{HF}_k}\leqslant 1.691\ 03$。

Galambos[20] 将物品尺寸的区间 $(0,1]$ 拓展到了 $(0,\alpha]$, 存在 h, $\alpha\in\left(\frac{1}{h+1},\frac{1}{h}\right]$, 所形成的箱子类型为 M 种, $M=k+h-1$, 且 $j<h$ 的那些 I_j-箱从不会使用。Galambos 给出了当 $\alpha\in\left(\frac{1}{h+1},\frac{1}{h}\right]$ 时, 装箱问题的所有将区间分成有限段的在线算法的渐近近似比的下界, 这个下界与 h 相关。

从渐近性能上来说, Harmonic 算法是最有效的一类在线算法, 有各种各样的变形和更加深入的分析。目前已知的在线装箱的渐近近似比处于 [1.542 780 9, 1.578 289 56] 区间内, 读者可参考文献 [21,22]。

降序 Fit 算法 前面提到的所有 Fit 算法均是按照物品的任意顺序来安排的, 一个物品到达后就立刻决定将其放在哪个箱子里, 不需要利用未来的信息。它们均为在线算法。在引出 NF 算法时, 我们知道, 物品的安排次序是非常重要的。如果我们可以先给物品排个次序, 那么什么样的顺序比较好呢? 依照我们的生活经验, 自然是先放尺寸大的物品会更省空间。于是便有了降序 Fit 算法 AFD:

- 将所有物品按照尺寸从大到小的顺序排列;
- 对排好顺序的物品使用 AF 算法。

我们得到 Next Fit Decreasing (NFD) 算法, First Fit Decreasing (FFD) 算法和 Best Fit Decreasing (BFD) 算法。Baker 和 Coffman[23] 证明了 NFD 算法的渐近近似比为 1.691。Garey 等人[24] 证明了对任意的 "物品尺寸均属于 $[1/5,1]$ 区间" 的实例 I, $\mathrm{FFD}(I)=\mathrm{BFD}(I)$。Johnson[4] 拓展了此结果, 证明了对任意的 "物品尺寸均属于 $[1/6,1]$ 区间" 的实例 I, $\mathrm{BFD}(I)\leqslant\mathrm{FFD}(I)$。

作为里程碑的工作, Johnson[4] 证明了对于装箱问题的任意实例 I, $\mathrm{FFD}(I)\leqslant\frac{11}{9}\mathrm{OPT}(I)+4$, $\mathrm{BFD}(I)\leqslant\frac{11}{9}\mathrm{OPT}(I)+4$。其证明过程相当复杂, 但其对装箱问题经典算法的研究带动了近似算法的发展。尤其是证明过程中构造权函数的分析方法, 为后续许多研究奠定了基础。大约过了一年, Johnson 等人[10] 就证明了 FFD 算法和 BFD 算法的渐近近似比恰为 $\frac{11}{9}$。Baker[25] 在 1985 年证明了对任意实例 I, $\mathrm{FFD}(I)\leqslant\frac{11}{9}\mathrm{OPT}(I)+3$。

此证明过程较之前简单了许多。1991 年, Yue[1] 更是将 FFD 算法的分析过程大大简化, 且得到了较前二者都更好的结果, 将尾项降为了 1。1997 年, Li 和 Yue[26] 将上述尾项分析到了 $\frac{7}{9}$, 并猜测精确的尾项为 $\frac{5}{9}$。又过了 10 年, Dósa[27] 将 FFD 算法的前述尾项改进到了 $\frac{6}{9}$, 且证明了尾项 $\frac{6}{9}$ 是紧的, 从而为 FFD 算法的分析画上圆满的句号。

通过非常简单的运算, Yue[1] 的结果其实已经表明 FFD 算法的绝对近似比为 1.5。我们知道, 除非 P=NP, 否则装箱问题不存在比 1.5 更好的绝对近似比[28]。事实上, 若装箱问题存在绝对近似比严格小于 1.5 的多项式时间近似算法, 则该算法能够在多项式时间内回答 "两个箱子是否能装下所有物品" 的判定问题, 与此问题的 NP 完全性相矛盾。从绝对近似比的角度看, FFD 算法已经是最好可能的算法了, 但其运行时间为 $O(n\log n)$。Zhang 等人[29] 利用非常简单的分拆组合技巧, 给出了绝对近似比为 1.5 的线性时间算法。至此, 我们看到了近年来装箱问题的巨大进展, 特别是经典 FF, BF 和 FFD 算法(渐近和绝对)近似比的完整刻画。

在 $\mathrm{P}\neq\mathrm{NP}$ 的假设下, 装箱算法的绝对近似比的研究已经结束。而对于渐近近似比的探讨似乎还会有很大的空间。是否存在渐近近似比好于 11/9 的近似算法? 其尾项会是多少?

注意到, 当物品的尺寸都非常小时, 使用前面所述的 Fit 算法就可以使渐近近似比充分接近于 1。故如何处理好比较大的物品才是改进渐近近似比的关键。

Fernandez de la Vega 和 Lueker[30] 在 1981 年给出了第一个 APTAS, 即证明了对任意 $\varepsilon>0$, 任意实例 I, 存在一簇线性时间算法 A_ε, 当 $\mathrm{OPT}(I)$ 充分大时, 总有 $A_\varepsilon(I)/\mathrm{OPT}(I)\leqslant 1+\varepsilon$, 即渐近近似比至多为 $1+\varepsilon$。Williamson 和 Shmoys[31] 将该算法作了更简洁的描述。首先, 设置阈值 $\gamma\in(0,1)$, 将所有尺寸大于 γ 的物品记为大物品, 否则为小物品。针对任意实例 I, 假设所有大物品共用了 ℓ 个箱子。在这 ℓ 个箱子的基础上, 对小物品使用 FF 算法。如果小物品完全装入到这 ℓ 个箱子中, 则箱子个数由大物品所确定。如若不然, 设最终所用箱子数为 $\ell_0+1>\ell$, 则前 ℓ_0 个箱子中的每一个所装物品尺寸都不小于 $1-\gamma$, 所以 $\mathrm{SIZE}(I)\geqslant(1-\gamma)\ell_0$, 其中 $\mathrm{SIZE}(I)$ 为 I 中所有物品尺寸总和。故而所使用的箱子数不超过 $\frac{1}{1-\gamma}\mathrm{SIZE}(I)+1$。

若想得到 APTAS, 借助前面的结论, 只需要给 γ 赋合适的值, 这里令 $\gamma=\varepsilon/2$, 可得 $\ell_0+1\leqslant\max\{\ell,(1+\varepsilon)\mathrm{OPT}(I)+1\}$。而后只需要单独考虑大物品该作何处理。将 I 中所有大于 $\varepsilon/2$ 的物品构成的实例记为实例 I'。每个箱子所装物品个数不超过 $2/\varepsilon$。此时如果我们同时可以限制物品的种类数也为常数的话, 产生的新实例便可以通过动态规划求得最优解。故可采用 "线性分组方案" 的技巧来压缩物品种类数且同时能保证装完 I' 中所有物品所用箱子总数不会较该部分最优解增加太多。

线性分组方案 将 I' 中所有物品按照尺寸从大到小的顺序从左到右依次排列, 基于参数 k_0, 从最大的 (最左端的) 物品开始将物品分组, 除最后一组外每组 k_0 个物品, 最后一组物品个数不超过 k_0。

舍入过程 分组之后, 将第一组物品删掉, 其余每组中物品尺寸均向上舍入到各组中尺寸最大的物品 (最左端的物品) 的尺寸。

作完删除和舍入操作后, 大物品仍保持大物品的属性, 记新实例为 I'', 动态规划处理 I'', I' 中所有物品可装入至多 $\mathrm{OPT}(I'')+k_0$ 个箱子里且 $\mathrm{OPT}(I'')+k_0\leqslant\mathrm{OPT}(I')+k_0$; 安排完所有的大物品后用 FF 算法安排 I 中小物品。取适当的 k_0 值, 这里取 $k_0=\lfloor\varepsilon\mathrm{SIZE}(I')\rfloor$, 这样既保证了大物品的种类数不超过 $4/\varepsilon^2$, 又得到了算法解所使用的箱子数不超过 $(1+$

$\varepsilon)\mathrm{OPT}+1$ 的结论。

以上算法过程对时间复杂度的要求无疑是非常高的。同时注意到, 对任意实例 I, $A(I)\leqslant\alpha\cdot\mathrm{OPT}(I)+k$, 当要求 k 为常数时, $(1+\varepsilon)\mathrm{OPT}+1$ 已为最好, α 和 k 的值均已减到最小, 尾项 1 不可能去掉, 因为装箱问题的绝对近似比不可能好于 1.5。当 k 不必为常数而为最优值的高阶无穷小 $k=o(\mathrm{OPT}(I))$ 时, 也许还有希望将 α 的值减小甚至降到 1。这对装箱问题而言也是非常有意义的贡献。下节将给出几个算法结果, 它们都将渐近近似比做到了 1, 尾项均为最优值的高阶无穷小且阶数由 $O(\log^2\mathrm{OPT})$ 到 $O(\log\mathrm{OPT}\cdot\log\log\mathrm{OPT})$ 再到 $O(\log\mathrm{OPT})$ 在一步步减小。

3　基于线性规划的舍入算法

线性规划舍入 (LP rounding), 是一类针对组合优化问题而言, 基于问题整数规划的线性规划松弛的分数解做舍入的算法。大致过程如下: 先写出问题的整数规划, 松弛为线性规划, 解此线性规划得到分数解, 针对此分数解利用一些方法获得 (舍入到) 整数解。

针对装箱问题的任意实例 I, 不妨设有 n 个物品, 最多使用 n 个箱子。用 i 代表物品, j 代表箱子。故 $i\in\{1,\cdots,n\}$, $j\in\{1,\cdots,n\}$。记 $\{1,\cdots,n\}=[n]$。装箱问题的 0-1 整数规划不难写出:

$$
\begin{aligned}
\min\quad & \sum_{j=1}^{n} y_j \\
\text{s. t.}\quad & \sum_{j=1}^{n} x_{ij}=1, && \forall i\in[n], \\
& \sum_{i=1}^{n} a_i x_{ij}\leqslant y_j, && \forall j\in[n], \\
& y_j\in\{0,1\}, && \forall j\in[n], \qquad (1)\\
& x_{ij}\in\{0,1\}, && \forall i,j\in[n], \qquad (2)
\end{aligned}
$$

其中, 若第 j 个箱子被使用了，则 $y_j=1$; 否则, $y_j=0$。若物品 i 被安排进第 j 个箱子里，则 $x_{ij}=1$; 否则, $x_{ij}=0$。

将问题的整数规划进行线性松弛: (1) (2) 两式分别变为 $y_j\in[0,1],\forall j\in[n]$ 和 $x_{ij}\in\mathbb{R},\forall i,j\in[n]$。然后对松弛解做舍入。试想, 如果我们得到的分数解每个都不为零, 但都是任意小, 于是按照所有非零变量均被我们入到 1 的舍入方法来获取整数解, 无疑是非常差的。如果采取更聪明一点的舍入方法, 近似比也绝对不会好于 2。事实上, 假设 n 个物品的尺寸均为 $\frac{1}{2}+\varepsilon$, 箱子容量为 1, 线性松弛的分数解的目标值为 $\frac{n}{2}+1$, 整数解最优值为 n, 这意味着该整数规划的整性间隙 (integrality gap) 至少是 2, 整性间隙是近似比的下界, 故由线性规划的分数解做任意舍入得到的整数可行解的目标值与整数最优值之间的比值至少是 2, 即近似比无论如何不会比 2 好。

为了减少直接对上述整数规划进行线性规划舍入带来的较大误差, 大家通常会采用另一种形式的整数规划 configuration IP (构型整数规划), 此时决策变量分数解代表的不再是物品在箱子里放了分数个, 而对应的是每类可行的装箱模式所用箱子有分数个。针对实例 $I=\{a_1,\cdots,a_n\}$, 用 m 代表物品的种类数, b_i 代表物品 a_i 的个数, s_i 代表物品 a_i

的尺寸。所有可行的装箱模式定义为 $\mathcal{P}=\{p\in\mathbb{Z}^m_{\geqslant 0}|\sum_{i=1}^m s_ip_i\leqslant 1\}$。若一种模式 p 内包含 a_i, 那么也说 p 内含有尺寸为 s_i 的槽 (slot)。写出该整数规划。

Configuration IP (构型整数规划)

$$\begin{aligned}\min\quad & \sum_{p\in\mathcal{P}} x_p\\ \text{s. t.}\quad & \sum_{p\in\mathcal{P}} p_ix_p\geqslant b_i, && \forall i\in[m],\\ & x_p\in\mathbb{N}, && \forall p\in\mathcal{P},\end{aligned}$$

其中x_p 表示解中所用装箱模式 p 的个数。

令 $\mathrm{OPT}_{\mathrm{IP}}(I)$ 表示实例 I 的整数最优解所对应的目标值, $\mathrm{OPT}_{\mathrm{LP}}(I)$ 表示上述整数线性规划松弛以后 ($x_p\geqslant 0$) configuration LP (构型线性规划) 的分数最优解对应的目标值。定义 $\mathrm{SIZE}(I)=\sum_{i=1}^m s_ib_i$。$\mathrm{OPT}_{\mathrm{LP}}(I)$ 有自然的下界和上界, 即 $\mathrm{SIZE}(I)\leqslant\mathrm{OPT}_{\mathrm{LP}}(I)\leqslant\mathrm{OPT}_{\mathrm{IP}}(I)$。

尽管此 configuration LP 的变量有指数多个, 但它是可以控制在一定的很小的误差范围内通过椭球法[32] 进行求解的。即将引入的三种算法均是基于 configuration LP 的解进行舍入的算法。

$(\mathrm{OPT}+O(\log^2\mathrm{OPT}))$ 的近似结果 对任意实例 I, 建立 configuration LP, 由线性规划的知识, 解方程组得基可行解 x^*, 其非零变量的个数不会超过 m, x^* 非零变量的整数部分 $\lfloor x^*\rfloor$ 所对应的物品即按照解中选择的装箱模式装, 分数部分 $x^*-\lfloor x^*\rfloor$ 非零变量的个数更不会超过 m, 故 m 个箱子一定能将分数部分所对应的物品全部装下; 记分数部分对应的所有物品的集合为 I'', 则这部分物品最多用 $2\mathrm{SIZE}(I'')$(NF 算法) 个箱子可全部装下; 综上, 在二者中选取较小的值作为 I'' 的解所对应的目标值, 则 I'' 中物品所用箱子数至多为 $\frac{m+2\mathrm{SIZE}(I'')}{2}=\frac{m}{2}+\mathrm{SIZE}(I'')$, 故 $\mathrm{OPT}_{\mathrm{IP}}(I)\leqslant\sum_{p\in\mathcal{P}}\lfloor x_p^*\rfloor+\frac{m}{2}+\mathrm{SIZE}(I'')$。需要注意的是, $\lfloor x^*\rfloor$ 是整数部分的分数最优解, $x^*-\lfloor x^*\rfloor$ 也是 I'' 部分的分数最优解, 否则 x^* 就不是原实例的分数最优解。因此, $\mathrm{SIZE}(I'')\leqslant\sum_{p\in\mathcal{P}}(x_p^*-\lfloor x_p^*\rfloor)$, $\mathrm{OPT}_{\mathrm{IP}}(I)\leqslant\sum_{p\in\mathcal{P}}\lfloor x_p^*\rfloor+\frac{m}{2}+\sum_{p\in\mathcal{P}}(x_p^*-\lfloor x_p^*\rfloor)=\sum_{p\in\mathcal{P}}x_p^*+\frac{m}{2}=\mathrm{OPT}_{\mathrm{LP}}(I)+\frac{m}{2}$, 即 m 越小越好。这里, m 为物品的种类数, 故设法减少物品的种类数应该是一条可行之路。

Karmarkar 和 Karp[33] 在 1982 年将装箱问题的渐近近似比做到了 1, 其尾项相较以前来看是非常大的, 对任意实例 I, 尾项为 $O(\log^2\mathrm{OPT}(I))$, 但其仍是 $\mathrm{OPT}(I)$ 的高阶无穷小。算法为: 首先对 I 中所有物品按照尺寸从大到小排列, 然后分组使 I 中物品分组后每组的尺寸之和介于 $[2,3)$ 之间, 舍弃每组中个数较 I 中前一组相比多余的排在后面 (相对较小) 的物品, 将每组剩下的物品尺寸增长到该组第一个物品的尺寸大小, 得到 I'。此刻 I' 中每组物品个数均与 I 中前一组物品个数相等。在上述过程中所有舍弃物品的尺寸总和不超过 $O(\log\mathrm{SIZE}(I))$。对 I' 构建 configuration LP, 得 x, 对 $\lfloor x\rfloor$ 所对应的物品 I_1 按照解中当前模式装箱; 对分数解部分所对应的物品 $I_2=I'-I_1$, 将 I_2 当成新的 I 重述上面过程。又因为 I' 中不同尺寸的物品个数不超过 $\frac{\mathrm{SIZE}(I)}{2}$, $\mathrm{SIZE}(I_2)$ 不超过 I' 中不同尺寸的物品个数 (线性规划中约束的个数), 故 $\mathrm{SIZE}(I_2)\leqslant\frac{\mathrm{SIZE}(I)}{2}$, 因而上述过程至多循环 $O(\log\mathrm{SIZE}(I))$ 次。所有轮次中丢掉的物品尺寸之和不超过 $O(\log\mathrm{SIZE}(I))\cdot\log\mathrm{SIZE}(I)$, 用 FF 算法去装它们至多需要 $O(\log^2\mathrm{SIZE}(I))$ 个箱子; $O(\log\mathrm{SIZE}(I))$ 次循环中所有整数部分所用的箱子总数不超过 $\mathrm{OPT}(I)$。因此整个过程中所用箱子数不超过 $\mathrm{OPT}(I)+O(\log^2\mathrm{SIZE}(I))\leqslant\mathrm{OPT}(I)+O(\log^2\mathrm{OPT}(I))$。

Karmarkar 和 Karp[33] 已经将渐近比做到了 1, 但此时尾项无疑是相当大的, 故如何在保持渐近比仍为 1 的前提下将尾项减小成为以后工作中学者们关注的焦点。

$(\mathrm{OPT}+O(\log \mathrm{OPT}\cdot \log\log \mathrm{OPT}))$ 的近似结果 Eisenbrand 等人[34] 指出 Karmarkar 和 Karp 算法的每一轮所用到的装箱模式均来自该轮初始时 configuration LP 的解, 且每个物品均被装进至少与自身尺寸一样大的槽里面。然而, 结合 [35] 和 [34] 两篇文章, 可知不存在只用上述简单的性质和向上舍入物品尺寸的方法就能将尾项降到 $\Omega(\log^2 n)$ 量级以下的算法。这种困难性同样对于包含 Karmarkar 和 Karp 算法在内的一类算法[36] 来说都是个瓶颈。

以下, 换种方式写出装箱问题的 configuration LP。首先, 对任意实例 I, 将其中所有不同种类的物品按照尺寸从大到小的顺序排列, 即 $1\geqslant s_1>s_2>\cdots>s_m>0$。

Configuration LP

$$\begin{aligned}\min\quad & \mathbf{1}^{\mathrm{T}}x=\sum_{p\in\mathcal{P}}x_p\\ \text{s. t.}\quad & A_ix\geqslant b_i, \qquad \forall i\in[m],\\ & x_p\geqslant 0, \qquad \forall p\in\mathcal{P},\end{aligned}$$

其中A 为装箱模式矩阵, A_i 表示 A 的第 i 行, 第 i 行第 p 列位置上的数字 A_{ip} 即表示第 i 种类型的物品在第 p 种装箱模式里的个数。

我们知道 Karmarkar 和 Karp[33] 的 $O(\log^2 \mathrm{OPT})$ 界的得出基于了如下事实: 线性方程组基本可行解中非零变量的个数不超过约束的个数。30 年后, Rothvoss[37] 使用差异理论 (discrepancy theory) 的技巧对尾项做出了改进, 即找到了一种在期望意义下多项式时间的随机算法将尾项降到了 $O(\log \mathrm{OPT}\cdot\log\log \mathrm{OPT})$。Hoberg 和 Rothvoss[38] 在 2017 年共同将尾项改进到了 $O(\log \mathrm{OPT})$。同样, 此结果也是在期望意义下的。两篇文章使用了相同的理论工具, 即差异理论中的部分着色引理[39]。

部分着色引理[39]　令 $x\in[0,1]^h$ 为起始点, $\delta>0$ 是任意的误差参数, 向量 $v_1,\cdots,v_d\in Q^h$, 参数 $\lambda_1,\cdots,\lambda_d\geqslant 0$ 且满足

$$\sum_{i=1}^{d}\mathrm{e}^{-\lambda_i^2/16}\leqslant\frac{h}{16},$$

则存在随机算法, 期望运行时间为 $O(\frac{(h+d)^3}{\delta^2}\log(\frac{hd}{\delta}))$, 找到向量 $y\in[0,1]^h$ 满足

(i) $|\{j\in[h]|y_j\in[0,\delta]\cup[1-\delta,1]\}|\geqslant\frac{h}{2}$,

(ii) $|v_iy-v_ix|\leqslant\lambda_i\cdot||v_i||_2$, $\forall i\in\{1,\cdots,d\}$。

由初始解 x 到新解 y, 若 $\delta>0$ 是很小的量, 当 $y_j\in[0,\delta]$, 将其舍到 0; 当 $y_j\in[1-\delta,1]$, 将其入到 1; 因此, 在目标函数值增量仅为常数的情况下, 可认为 y 至少有一半的分量落在了 0 或 1。

可以看出, 部分着色引理是非常好的线性规划舍入工具: 给定任意分数解 x, 想要获得一个整数解 y, 这时我们可以在满足误差限制 $(\sum_i \mathrm{e}^{-\lambda_i^2/16}\leqslant\frac{h}{16})$ 的条件下增加线性约束。若 $\lambda_i=0$, 则增加的约束为 $v_iy=v_ix$, 此时对误差项的贡献为 $\mathrm{e}^0=1$; 若 $\lambda_i\neq 0$, 则增加的约束为 $|v_iy-v_ix|\leqslant\lambda_i\cdot||v_i||_2$, 此时对误差项的贡献为 $\mathrm{e}^{-\lambda_i^2/16}<1$。即 $|\lambda_i|$ 越大, 误差限制条件越容易满足。

于是, Rothvoss[37] 算法的大致框架就有了: 针对 configuration LP 的解, 将其所有非零分量构成的新向量取出得 $x \in \mathbb{R}^m_{\geqslant 0}$, 整数部分直接按照解中模式装箱。对分数部分, 运行部分着色引理, 使得到的 y^1 至少有一半的分量变为了整数; 针对 y^1 中的分数分量部分, 运行部分着色引理得 y^2, 其中又会有至少一半的分量变为了整数; 如此循环做下去直至所有的分量都为整数, 即找到了整数解 y^t, $t \leqslant \log m$。

那么到底该如何选择 v_i 和 λ_i 呢? 我们先来看看给定 x 以后, y 会具备什么样的性质。注意到此刻所有物品均已按照尺寸从大到小的顺序排列好。如果 y 中提供的尺寸至少为 s_i 的槽的数量不少于尺寸至少为 s_i 的物品的总个数, 且对任意的 i 都成立, 则 y 是可行解; 否则存在一些 i, y 中提供的尺寸至少为 s_i 的槽的数量少于尺寸至少为 s_i 的物品的总个数, 则取二者数量差的绝对值的最大值记为 M, 将尺寸最大的 M 个物品单独装箱, y 是剩下物品的可行解。因此, 我们一定想尽力减少 M, 即减少 $|\sum_{j\leqslant i} A_j(x-y)|$。

仅需考虑所有 $\left[\frac{1}{m}, 1\right]$ 之间的物品, 可分为 $\log m$ 个尺寸区间, 每个尺寸区间为 $[\frac{1}{k}, \frac{2}{k}]$, k 为 2 的幂次。对每个尺寸区间来说, 区间里的物品可以经由 $\log m$ 次部分着色引理找到该部分整数解, 所有尺寸区间同时执行相同的操作。经过粗略地分析, 尾项不超过 $\log^2 m$ 乘上每次使用部分着色引理增加的误差。当这个误差为常数时, 尾项也仅能与 Karmarkar 和 Karp 得到的界一样好。以下先用部分着色引理分析一下这个误差。通过观察, 定义 $v_G = \sum_{i\in G} A_i$, 其中 $G \subseteq [m]$。

为简单起见, 我们假设所有的物品尺寸均落在 $[\frac{1}{k}, \frac{2}{k}]$ 之间。解 x 已知时, 系数矩阵 A 已固定。从上到下取连续的几排构成一组 G_1, 使得 $||v_{G_1}||_1 \approx 100k$; 紧接着, 在剩下的 A 中从上到下取连续的几排构成一组 G_2, 同样使得 $||v_{G_2}||_1 \approx 100k$; 依次进行下去, 直到取到第 i ($\forall i$) 排。我们得到一些组 $G_1, \cdots, G_{r-1}$ 和一个 $||v_{G_r}||_1 < 100k$ 的剩余组 G_r。选定了 v_{G_j} 后, 不妨先尝试一下若令所有 $j \leqslant r-1$ 的 λ_{G_j} 都为零会有什么结论。

想要针对所有 $G_1, \cdots, G_{r-1}$ 所对应的 v, λ 使用部分着色引理, 首先要验证误差限制条件: 因为物品尺寸均在 $[\frac{1}{k}, \frac{2}{k}]$ 区间内, 故 A 的每一列至多有 k 个物品, 组数 $r-1 \leqslant \frac{mk}{100k} = \frac{m}{100}$, 故 $(r-1)\cdot \mathrm{e}^{0^2/16} \leqslant \frac{m}{100} \leqslant \frac{m}{16}$。应用引理可找到 y, 满足 $v_{G_j}(x-y) = 0, \forall j \in [r-1]$。于是有

$$\left|\sum_{j\leqslant i} A_j(x-y)\right| = \left|\sum_{j=1,\cdots,r} v_{G_j}(x-y)\right| = \left|v_{G_r}(x-y)\right| \leqslant 100k,$$

其中最后一个不等号成立是因为 $x-y$ 的每个分量的绝对值不超过 1。

也就是说, 上式对于任意的 i 总是成立的, 即意味着上述找到的 y 是 $G_2, \cdots, G_r$ 中物品的可行解。然后只需单独对 G_1 中物品装箱, 已知 G_1 中大约有 $100k$ 个物品, 且每个物品的尺寸均不超过 $\frac{2}{k}$, 故该部分所使用箱子数不超过 400 个。解 y 中所用箱子数量我们还没有估计, 但容易看出, 在前述基础上添加 $v_{\mathrm{obj}} = (1, \cdots, 1), \lambda_{\mathrm{obj}} = 0$, 误差限制条件可仍然满足且得到 $\mathbf{1}^{\mathrm{T}}x = \mathbf{1}^{\mathrm{T}}y$。因此, 运用一次部分着色引理所带来的目标值增量为常数。到此, 我们真正利用部分着色引理得到了 $\log^2 m$ 尾项, 这里, m 为物品的种类数。尾项 $\log^2 m$ 是可以通过一些简单的技巧转化为 $\log^2 \mathrm{OPT}$ 的。

然而, 上面的讨论还是没有对 $\log^2 \mathrm{OPT}$ 的结果做出改进, 于是我们想若一次部分着色引理可以带来的误差为次常数, 或者说比常数的阶小的数, 这样还有希望对 $\log^2 \mathrm{OPT}$ 的结果做出改进。回看上述过程, 部分着色引理并没有发挥它的强大作用: $v_{G_j}(x-y) = 0, \forall j \in [r-1]$ 保证了每个组内物品总个数与 y 提供的槽的个数相等, 但有可能 y 中含有的尺寸较小的槽的个数比较多而使得组里部分大物品没有较大空间的槽可以安放。因此

考虑增添一些 v_i, λ_i 以改善这个情况。

对任意一个 $||v_{G_j}||_1 \approx 100k$ 的 G_j, 创建具有包含关系的集合序列: $G_{j_1} \subseteq G_{j_2} \subseteq \cdots \subseteq G_{j_{\log^2 m}} = G_j$, 其中, $||v_{G_{j_{l+1}} \setminus G_{j_l}}||_1 \approx \frac{1}{\log^2 m} \cdot 100k$。

因此, 对任意的 i, 剩余组 G_r 可分为一个极大的子组 $G_{r_s} \subseteq G_r$ 加上 G_{r_r}, 其中 $||v_{G_{r_s}}||_1 \approx s \cdot \frac{1}{\log^2 m} \cdot 100k$, $||v_{G_{r_r}}||_1 < \frac{1}{\log^2 m} \cdot 100k$。令 $\lambda_{G_j} = 0, j \leqslant r-1$, $\lambda_{G_{r_s}} = \log\log m$, 针对 $G_1, \cdots, G_{r-1}, G_{r_s}$ 所对应的 v, λ, 易验证误差限制条件成立, 使用部分着色引理得 y, $v_{G_j}(x-y) = 0, \forall j \in [r-1]$; $|v_{G_{r_s}}(x-y)| \leqslant \lambda_{G_{r_s}} ||v_{G_{r_s}}||_2$, 故有 $|\sum_{j \leqslant i} A_j(x-y)| = |\sum_{j=1,\cdots,r} v_{G_j}(x-y)| = |v_{G_{r_s}}(x-y) + v_{G_{r_r}}(x-y)| \leqslant \log\log m \cdot ||v_{G_{r_s}}||_2 + \frac{1}{\log^2 m} \cdot 100k$。

可以看出，十分关键的一点在于子组 G_{r_s} 的 $||v_{G_{r_s}}||_2$ 是否能够控制在较小的界内。注意到

$$||v_{G_{r_s}}||_2 \leqslant \sqrt{||v_{G_{r_s}}||_1 \cdot ||v_{G_{r_s}}||_\infty} = \sqrt{\frac{||v_{G_{r_s}}||_\infty}{||v_{G_{r_s}}||_1}} \cdot ||v_{G_{r_s}}||_1,$$

其中 $||v_{G_{r_s}}||_1 \leqslant O(k)$, 故只需控制 $\frac{||v_{G_{r_s}}||_\infty}{||v_{G_{r_s}}||_1}$。观察可得, 当一个装箱模式里有许多尺寸相同的物品或者尺寸相近的物品时, $\frac{||v_{G_{r_s}}||_\infty}{||v_{G_{r_s}}||_1}$ 仍可达到一个常数量级。此时还是没能改变 $\log^2 \mathrm{OPT}$ 的界。因此, 想要改变这个糟糕的情况, 可将尺寸相同的物品进行粘连 (gluing)。

假设我们有一个装箱模式 p, $x_p = \frac{1}{q}$, q 为正整数。p 中有许许多多尺寸相同的物品, 假定其数量为 $k \cdot q$, 这就意味着 p 已经能够装下 k 个这种物品了。此时考虑将这种物品每 k 个粘连成一个新的物品其尺寸为原来的 k 倍, 加到输入中, 同时删除被粘连的原输入中的物品。与此同时, 将分数解 x 中 k 个相同的槽粘连成一个大的槽, 得到新解 x', 这不改变可行性与目标值, 且任何新实例的整数解都是原实例的整数解。

对任意的装箱模式 p, 可通过一些技巧使所有的 x_p 为 $\frac{1}{\mathrm{poly}(\log m)}$。令 $\varepsilon = \frac{1}{\log^4 m}$, 对每个 p 和 i, 如果我们做到 $p_i \leqslant \varepsilon \cdot A_i x$, 则 $\frac{||v_{G_{r_s}}||_\infty}{||v_{G_{r_s}}||_1}$ 就会不超过 ε。否则, 存在 p 和 i, 不满足 $p_i \leqslant \varepsilon \cdot A_i x$。首先用类似于 Karmarkar 和 Karp 的分组方法进行分组 (grouping), 以使每个物品类型要么不存在了, 要么该物品类型总尺寸至少是 ε。这个过程费用不超过 $O(\varepsilon \cdot \log m)$。然后, 检验哪个 p 和 i 不满足条件, 运用粘连技巧, 将一连串的小物品粘为一个物品。可以验证, 新得到的物品的尺寸至少为 ε^3 量级, 是 $\mathrm{poly}(\log m)$ 量级的。不可能再被粘连第二次。因为其实需要粘连的物品一般都是小物品, 在这里我们就假设只针对 $\mathrm{poly}(\log m)$ 量级以下的物品才做粘连操作。也只需要对 $\mathrm{poly}(\log m)$ 量级以下的物品 i 和任意 p 做到 $p_i \leqslant \varepsilon \cdot A_i x$。

经过粘连后, 若剩下的每种物品总尺寸均至少为 $\frac{\varepsilon}{2}$, 则可验证对所有的小于 $\mathrm{poly}(\log m)$ 的物品类型 i, $p_i \leqslant \varepsilon \cdot A_i x$ 成立。否则, 去掉 $\mathrm{poly}(\log m)$ 量级下总尺寸至少为 $\frac{\varepsilon}{2}$ 的物品类型, 针对所有剩下来物品, 继续执行分组、粘连操作, 直至对于所有 p 和 i, $p_i \leqslant \varepsilon \cdot A_i x$ 总成立。不难看出, 每次循环后需要考虑的总尺寸较上次至少减少一半, 故该循环至多执行 $\log m$ 次。同时, 由此产生的费用相应为 $\log m \cdot O(\varepsilon \cdot \log m) = O(\frac{1}{\log^2 m})$。

到此, 算法框架逐步清晰了: 针对 configuration LP 的解的分数部分 x; 对 x 做舍入使得 x_p 为 $\frac{1}{\log^4 m}$ 的整数倍; 应用粘连技巧; 运行部分着色引理使得至少一半的分量变为整数得到新解, 其分数分量部分记为 y^1; 针对 y^1 做舍入使得 y^1_p 为 $\frac{1}{\log^4 m}$ 的整数倍; 应用粘连技巧; 运行部分着色引理使得至少一半的分量变为整数得到新解, 其分数分量部分记为 y^2; 循环上述过程直到得到整数解。该过程至多持续 $\log m$ 次。

我们将 $\left[\frac{1}{m}, 1\right]$ 之间的物品, 分进了至多 $\log m$ 个尺寸区间内。对于 $\{[\frac{1}{k}, \frac{2}{k}] | k >$

$\mathrm{poly}(\log m)\&k = 2^\ell, \ell \in [\log m]\}$ 中的每个区间而言, 经过分组、粘连后使许多同类型的小物品连在一起成为大物品从而逃离了它们原本所在的区间继而转入了 $\{[\frac{1}{k}, \frac{2}{k}]|k \leqslant \mathrm{poly}(\log m)\&k = 2^\ell, \ell \in [\log m]\}$ 中的某个区间, 转移费用为 $O(\frac{1}{\log^2 m})$。$\{[\frac{1}{k}, \frac{2}{k}]|k > \mathrm{poly}(\log m)\&k = 2^\ell, \ell \in [\log m]\}$ 中的每个区间所剩下的为数不多的小物品均满足 $\frac{||v_{G_{r_s}}||_\infty}{||v_{G_{r_s}}||_1}$ 特别小, 因此应用部分着色引理得到可行解产生的费用均特别小, 为 $O(\frac{1}{\mathrm{poly}(\log m)})$。这些区间产生的总费用不超过 $O(1)$。

对于 $\{[\frac{1}{k}, \frac{2}{k}]|k \leqslant \mathrm{poly}(\log m)\&k = 2^\ell, \ell \in [\log m]\}$ 的每个区间来说, 做法与最开始得到 $\log^2 m$ 的讨论相同, 每个区间所产生的费用均为常数 $O(1)$。这些区间产生的总费用不超过 $O(\log\log m)$。

综上所述, 每个区间分别同时运行 $\log m$ 次部分着色引理, 得到最终的可行解所产生的附加总费用 (尾项) 不超过 $O(\log m \cdot \log\log m)$, 进而通过一些简单的技巧转化为 $O(\log \mathrm{OPT} \cdot \log\log \mathrm{OPT})$。至此, 文 [37] 结果的思路分析已全部完成。它将物品分为大物品和小物品, 对大物品的处理结果与 Karmarkar 和 Karp 算法针对大物品所得结果相同, 对小物品实施了分组、粘连技术使得部分小物品合为大物品, 从而降低了小物品对全局误差的影响。前面已经提到过, 2017 年 Hoberg 和 Rothvoss[38] 共同将尾项改进到了 $O(\log \mathrm{OPT})$。以下将大致提一下文章的算法框架。

$(\mathrm{OPT} + O(\log \mathrm{OPT}))$ 的算法结果 [37] 和 [38] 两篇文章研究思路类似: 从线性规划松弛解 (分数向量) 出发, 经过一系列构建过程使获得的参数满足部分着色引理的条件, 进而可获得至少有一半的分量变为整数的新解, 同时, 目标函数值在此过程中会有所增加 (文[37] 和 [38] 中目标函数值分别至多增加 $O(\log\log \mathrm{OPT})$ 和 $O(1)$)。而我们的目的是要得到整数可行解, 所以最多进行对数次上述过程即可实现。相应地, 目标函数值累加对数次。

观察到, 从 $O(\log^2 \mathrm{OPT})$ 到 $O(\log \mathrm{OPT} \cdot \log\log \mathrm{OPT})$, 在运用部分着色引理过程中 λ 值的选取从单一的 0 变为了 0 和 $\log\log m$。分组细化且不同分组对应的 λ 值不同, 从而使算法的每一次循环所带来的增量变小, 结果更好。于是引入多种类的 λ 值也许能得到更好的结果。文章 [38] 中分组更加精细了, λ 值的种类也更加丰富了。

它首先写出该问题的 configuration LP, 解线性规划得 $x \in \mathbb{R}^m_{\geqslant 0}$, 整数部分所对应的物品直接按照解给出的模式装箱, 后只需单独考虑分数部分的物品。从前面的分析可知若能将原实例转化成只需考虑较少种类物品的同时控制目标值增量, 就能得到比较不错的结果。注意到在文 [38] 中, 作者不是考虑直接把物品装进箱子里而是先制造一些固定的容器 (container) 将物品放进这些容器内, 而后再将这些容器放进给定的装箱模式里。这里, 所有可行的容器集合定义为 $\mathcal{C} = \{C|s(C) := \sum_{i=1}^m C_i s_i \leqslant 1\}$, 其中 C_i 表示容器 C 中尺寸为 s_i 的物品的数量; 所有可行的装箱模式集合定义为 $\mathcal{P} := \{p \in \mathbb{Z}^{\mathcal{C}}_{\geqslant 0} | \sum_{C\in\mathcal{C}} p_C \cdot s(C) \leqslant 1\}$, 其中 p_C 表示模式 p 中容器 C 的数量。

在第一阶段把物品放到给定的容器内时有可能出现一些物品无处安放的情况, 同样在第二阶段容器也有可能无法全部放进给定装箱模式的箱子里。这时候把所有被剩下的物品和容器按照此刻的状态放在一起另外考虑。通过每一步未装进容器的物品总尺寸和未装进固定装箱模式里的容器总尺寸的不断变化, 最终构造出目标函数值不超过 $\mathrm{OPT} + O(\log \mathrm{OPT})$ 的可行解。

算法框架与文 [37] 基本一致: 针对 configuration LP 的解的分数部分 x; 构建关于容器的系数矩阵 A; 运行部分着色引理使得至少一半的分量变为整数得到新解, 其分数分量

部分记为 y^1; 针对 y^1 构建关于容器的系数矩阵 A; 再运行部分着色引理使至少一半的分量变为整数得到新解, 其分数分量部分记为 y^2; 循环该过程直到得到整数解。该循环至多持续 $\log m$ 次。上述每次重构关于容器的系数矩阵 A 的过程, 费用增量不超过 $O(1)$; 应用一次部分着色引理的费用增量不超过 $O(1)$, 于是附加总费用 (尾项) 不超过 $O(\log m)$, 进而将尾项控制在 $O(\log \mathrm{OPT})$ 内。

文[38] 和文 [37] 相比, 都使用了部分着色引理, 算法框架大致相同, 且部分结论相同, 部分证明过程也相同, 但物品的分组方式不同使得构造出的部分着色引理的输入大不相同, 具体算法和分析不再给出。值得注意的是, 文 [38] 构造的输入参数的多样化使得每次循环所带来的目标函数值增量为常数。这已经是利用部分着色引理所能做到的最好结果了, 除非我们能对部分着色引理[39] 做出改进: 如, 从分数解 x 到整数解 y 的旅途上不用再漂洋过海, 而只需行一衣带水路, 譬如 $\log\log \mathrm{OPT}$ 或是其他的 $o(\log \mathrm{OPT})$。

除利用部分着色引理之外, 对于经典的一维装箱问题在保持渐近近似比的同时还没有更好的改进尾项的方法。大家接连发问: 尾项能否降到常数甚至 1 呢? 2010 年, Jansen 和 Solis-Oba[40] 在物品的种类数为固定常数时找到了 $\mathrm{OPT}+1$ 的近似算法; Goemans 和 Rothvoss[41] 在 2014 年证明了当物品的种类数为固定常数时, 装箱问题存在多项式时间的最优算法。但对于一般情况, 还未能攻克。

4　装箱问题的变形问题

经典的一维装箱问题的研究是各类纷繁复杂的装箱问题的研究基础, 所以针对装箱问题的变种问题, 在这也仅限于在一维离线情况下进行讨论。

在经典的装箱问题中, 所有的箱子容量是一定的, 但现实生活生产中箱子容量有大有小也许更可能带来较高的收益, 故箱子容量不固定的变形是非常具有研究价值的。称这种问题为变尺寸装箱问题。变尺寸装箱问题最早由 Friesen 和 Langston[42,43] 开始研究。它是 NP 困难的, 因为经典的装箱问题是它的特例, 它自然可以由经典的装箱问题归约过来。

当箱子容量不固定时, 常见有以下几种问题模型:

- 目标是最小化所用箱子的容量和 (容量和费用成比例时)[43]
- 目标是最小化所用箱子的费用和 (容量和费用无关时)[44]
- 物品之间有冲突: 特定物品不能放在同一个箱子里[45-47]

除了上述所说变形外, 还有一类涉及到博弈中机制设计的装箱问题。这时, 可以换一个角度来考虑: 在经典的装箱问题中, 只有一个决策者, 他只在乎全局的目标而不关心每个物品自身的利益。现在假设在原有问题基础上, 每个物品都被一个玩家操控, 且假设所有玩家都是理性 (自私) 的, 则每个玩家 (物品) 都会选择最大化自身的利益。这就引出了自私装箱 (selfish bin packing) 的概念。这里, 假设每个箱子费用均为 1, 箱子里面的物品需要根据一定的规则分摊箱子的费用, 每个物品都想让自身的利益最大化, 也就是说每个物品都愿意选择让自己付出最小费用的箱子。机制设计者的目标就是要设计出一种费用分摊机制使每个物品在处于稳定状态的同时最小化社会总费用。

稳定状态是指没有物品愿意移动到另一个箱子而获取更高的利益, 这个状态也就是我们平常所说的纳什均衡。在这个阶段使用 PoA (Price of Anarchy)[48] 这个概念来描述机制的性能, 它定义为最坏的纳什均衡对应的目标值和最优值之间的比率。PoA 大于等

于 1 且越小越好[49]。

5 总 结

装箱问题和与生产计划、资源配置相关的一类排序问题有非常紧密的联系。定义一个特殊的排序问题: 给定若干台用以加工工件的机器, 机器之间没有区别, 给定 n 个需要加工的工件, 每个工件有自己的加工时长, 释放时间均为 0, 目标是要用尽可能少数量的机器使得所有的工件在给定的统一截止时间之前被加工完成。将这个问题映射到装箱问题上, 机器相当于箱子, 带加工时长的工件相当于带尺寸的物品, 统一的截止日期相当于一致的箱子容量。故上述特殊的排序问题就是经典的装箱问题。对装箱问题的深入研究势必会带动诸如排序等一大类组合优化问题的研究。

本文重点介绍了装箱问题的经典 Fit 算法和线性规划舍入算法, 对近几年新结果的研究思路作了一些分析。装箱问题高维情形及在线情况本文未过多涉及, 装箱的变种问题的研究结果也只列出了比较基本的几篇文献, 在上述多方面还存在许多值得研究的问题。特别地，在经典装箱领域里, 最重要的未解问题为：一维装箱问题是否存在绝对近似算法, 即是否存在常数 c 和多项时间算法 A, 对所有的实例 I, 有 $A(I) \leqslant \mathrm{OPT}(I) + c$?

参 考 文 献

[1] Yue M Y. A simple proof of the inequality FFD $(L) \leqslant 11/9\mathrm{OPT}(L) + 1, \forall L$ for the FFD bin-packing algorithm [J]. *Acta Mathematicae Applicatae Sinica*, 1991, **7**(4): 321-331.

[2] Eisemann K. The trim problem [J]. *Management Science*, 1957, **3**(3): 279-284.

[3] Ullman J D. The performance of a memory allocation algorithm [R]. Princeton: Princeton University, 1971.

[4] Johnson D S. Near-optimal bin packing algorithms [D]. Cambridge: Massachusetts Institute of Technology, 1973.

[5] Garey M R, Johnson D S. Approximation algorithms for bin packing problems: A survey [M]// *Analysis and Design of Algorithms in Combinatorial Optimization*, New York: Springer, 1981: 147-172.

[6] Coffman E G, Garey M R, Johnson D S. Approximation algorithms for bin-packing-an updated survey [M]//*Algorithm Design for Computer System Design*, New York: Springer, 1984: 49-106.

[7] Coffman E G, Garey M R, Johnson D S. Approximation algorithms for bin packing: A survey [J]. *Approximation Algorithms for NP-hard Problems*, 1996: 46-93.

[8] Coffman E G, Csirik J, Galambos G, et al. Bin packing approximation algorithms: Survey and classification [M]. *Handbook of Combinatorial Optimization*, 2013: 455-531.

[9] Johnson D S. Fast algorithms for bin packing [J]. *Journal of Computer and System Sciences*, 1974, **8**(3): 272-314.

[10] Johnson D S, Demers A, Ullman J D, et al. Worst-case performance bounds for simple one-dimensional packing algorithms [J]. *SIAM Journal on Computing*, 1974, **3**(4): 299-325.

[11] Garey M R, Graham R L, Johnson D S, et al. Resource constrained scheduling as generalized bin packing [J]. *Journal of Combinatorial Theory, Series A*, 1976, **21**(3): 257-298.

[12] Dósa G, Sgall J. First fit bin packing: A tight analysis [C]//*Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2013: 538-549.

[13] Dósa G, Sgall J. Optimal analysis of best fit bin packing [C]//*Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, 2014: 429-441.

[14] Csirik J, Imreh B. On the worst-case performance of the NkF bin-packing heuristic [J]. *Acta Cybernetica*, 1989, **9**(2): 89-105.

[15] Mao W. Tight worst-case performance bounds for next-k-fit bin packing [J]. *SIAM Journal on Computing*, 1993, **22**(1): 46-56.

[16] Mao W. Besk-k-Fit bin packing [J]. *Computing*, 1993, **50**(3): 265-270.

[17] Zhang G, Yue M. Tight performance bound of AFB_k bin packing [J]. *Acta Mathematicae Applicatae Sinica*, 1997, **13**(4): 443-446.

[18] Csirik J, Johnson D S. Bounded space on-line bin packing: Best is better than first [J]. *Algorithmica*, 2001, **31**(2): 115-138.

[19] Lee C C, Lee D T. A simple on-line bin-packing algorithm [J]. *Journal of the ACM*, 1985, **32**(3): 562-572.

[20] Galambos G. Parametric lower bound for on-line bin-packing [J]. *SIAM Journal on Algebraic Discrete Methods*, 1986, **7**(3): 362-367.

[21] Balogh J, Békési J, Dósa G, et al. A new lower bound for classic online bin packing [J]. *Algorithmica*, 2021, **83**(7): 2047-2062.

[22] Balogh J, Békési J, Dósa G, et al. A new and improved algorithm for online bin packing [C]//*Proceedings of the 26th European Symposium on Algorithms (ESA)*, 2018: 1-5:14.

[23] Baker B S, Coffman E G. A tight asymptotic bound for next-fit-decreasing bin-packing [J]. *SIAM Journal on Algebraic Discrete Methods*, 1981, **2**(2): 147-152.

[24] Garey M R, Graham R L, Ullman J D. Worst-case analysis of memory allocation algorithms [C]//*Proceedings of the 4th Annual ACM Symposium on Theory of Computing (STOC)*, 1972: 143-150.

[25] Baker B S. A new proof for the first-fit decreasing bin-packing algorithm [J]. *Journal of Algorithms*, 1985, **6**(1): 49-70.

[26] Li R, Yue M Y. The proof of FFD $(L) \leqslant 11/9\mathrm{OPT}\ (L) + 7/9$ [J]. *Chinese Science Bulletin*, 1997, **42**(15): 1262-1265.

[27] Dósa G. The tight bound of first fit decreasing bin-packing algorithm is FFD $(I) \leqslant 11/9\mathrm{OPT}\ (I) + 6/9$ [C]//*International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, 2007: 1-11.

[28] Garey M R, Johnson D S. *Computers and Intractability: A Guide to the Theory of NP-Completeness* [M]. New York: WH Freeman and Company, 1979.

[29] Zhang G, Cai X, Wong C K. Linear time-approximation algorithms for bin packing [J]. *Operations Research Letters*, 2000, **26**(5): 217-222.

[30] Fernandez de la Vega W, Lueker G S. Bin packing can be solved within $1+\varepsilon$ in linear time [J]. *Combinatorica*, 1981, **1**(4): 349-355.

[31] Williamson D P, Shmoys D B. *The Design of Approximation Algorithms* [M]. Cambridge: Cambridge University press, 2011.

[32] Grötschel M, Lovász L, Schrijver A. The ellipsoid method and its consequences in combinatorial optimization [J]. *Combinatorica*, 1981, **1**(2): 169-197.

[33] Karmarkar N, Karp R M. An efficient approximation scheme for the one-dimensional bin-packing problem [C]//*Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, 1982: 312-320.

[34] Eisenbrand F, Pálvölgyi D, Rothvoss T. Bin packing via discrepancy of permutations [C]//*Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011: 476-481.

[35] Newman A, Neiman O, Nikolov A. Beck's three permutations conjecture: A counterexample and some consequences [C]//*Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, 2012: 253-262.

[36] Eisenbrand F, Pálvölgyi D, Rothvoss T. Bin packing via discrepancy of permutations [J]. *ACM Transactions on Algorithms*, 2013, **9**(3): 1-15.

[37] Rothvoss T. Approximating bin packing within O(log OPT∗log log OPT) bins [C]//*Proceedings of the 54th Annual Symposium on Foundations of Computer Science (FOCS)*, 2013: 20-29

[38] Hoberg R, Rothvoss T. A logarithmic additive integrality gap for bin packing [C]//*Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2017: 2616-2625.

[39] Lovett S, Meka R. Constructive discrepancy minimization by walking on the edges [J]. *SIAM Journal on Computing*, 2015, **44**(5): 1573-1582.

[40] Jansen K, Solis-Oba R. An OPT + 1 algorithm for the cutting stock problem with constant number of object lengths [C]//*Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2010: 438-449.

[41] Goemans M X, Rothvoss T. Polynomiality for bin packing with a constant number of item types [C]//*Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014: 830-839.

[42] Friesen D K, Langston M A. A storage-size selection problem [J]. *Information Processing Letters*, 1984, **18**(5): 295-296.

[43] Friesen D K, Langston M A. Variable sized bin packing [J]. *SIAM Journal on Computing*, 1986, **15**(1): 222-230.

[44] Epstein L, Levin A. An APTAS for generalized cost variable-sized bin packing [J]. *SIAM Journal on Computing*, 2008, **38**(1): 411-428.

[45] Jansen K, Öhring S. Approximation algorithms for time constrained scheduling [J]. *Information and Computation*, 1997, 132(2): 85-108.

[46] Jansen K. An approximation scheme for bin packing with conflicts [J]. *Journal of Combinatorial Optimization*, 1999, **3**(4): 363-377.

[47] Epstein L, Levin A. On bin packing with conflicts [J]. *SIAM Journal on Optimization*, 2008, **19**(3): 1270-1298.

[48] Koutsoupias E, Papadimitriou C. Worst-case equilibria [C]//*Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, 1999: 404-413.

[49] Zhang C, Zhang G. Cost-sharing mechanisms for selfish bin packing [C]//*Proceedings of the 11th International Conference on Combinatorial Optimization and Applications (COCOA)*, 2017: 355-368.