受启动空间约束的装箱问题*

顾晓东, 许胤龙, 陈国良, 黄刘生

(国家高性能计算中心(合肥),安徽 合肥 230027);
(中国科学技术大学 计算机科学与技术系,安徽 合肥 230027)
E-mail: gxd@mail.ustc.edu.cn
http://www.nhpcc.ustc.edu.cn

摘要: 提出了一种带有启动空间的约束装箱问题(start-up bin packing problem,简称 SBPP),即不同类型的物品放入同一箱子中需要一个启动空间.该问题在工作分配、任务调度和日常生活中的包装等问题中有着广泛的应用背景.给出了一个求解 SBPP 的线性脱线算法 C-NF,其最坏情况渐近性能比为 2,与启动空间的大小无关.对该算法的平均性能进行了实验分析.另外,还分析了 SBPP 的在线特性,指出大量的经典在线装箱算法应用于 SBPP 都不存在确定的最坏情况渐近性能比,也给出了一种具有确定的最坏情况渐近性能比的在线算法.

关 键 词: 装箱问题;组合优化;近似算法;最坏情况渐近性能比;平均性能比
中图法分类号: TP301 文献标识码: A

本文提出一种带约束的装箱问题.给定 n 个物品的序列 $L_n=(a_1,a_2,\ldots,a_n)$,物品 $a_i(1\le i\le n)$ 的大小 $s(a_i)\in(0,1-\delta)(0<\delta<1$ 是常数)、所属类型 $c(a_i)$.要求将这些物品装入单位容量的箱子 $B_1,B_2,\ldots,B_m$ 中,使得每个箱子中的物品大小之和不超过 1,并使所使用的箱子数目 m 最小.而当箱子中第 1 次放入属于某种类型 γ 的物品 a 时,a 除了占用 $s(a)$的容量以外,还要占用额外容量 δ 作为类型 γ 的启动放置空间(其作用可能是类型的启动预处理、保护措施等),即 a 共占用 $s(a)+\delta$ 的容量,而以后再向这个箱子中放入类型 γ 的物品时,则不再需要额外的启动放置空间.称这种问题为带有启动约束的装箱问题(start-up bin packing problem,简称 SBPP).

带有启动约束的装箱问题是经典一维装箱问题(bin packing problem,简称 BPP)的自然推广,实际上当取 $\delta=0$ 或者令所有物品属于同一类型时,SBPP 就退化为 BPP.在大量的实际应用中,装箱问题都带有如上所述的这种启动约束:

(1) 在计算机科学应用中,这种情况随处可见.例如,一个软件开发小组要在规定时间内完成一项任务,系统分析员把任务划分成各个子任务.由于每个子任务要求的背景知识不同,而每种背景知识都需要花费程序员很多的时间去学习和掌握.系统分析员在分派任务的时候要考虑到各背景知识的学习时间(即 SBPP 中的启动空间),不合理的分派将导致各程序员忙于了解各种背景知识而贻误工期.

(2) 在日常生活的包装问题中,为了减少在运输过程中的振动所造成的损失,往往要准备足够多的泡沫垫把同一箱子中不同材料的物品隔开,而相同材料的物品之间由于其内在的亲合性可以省去这种浪费.在这个应用中,泡沫垫所占用的箱子容量即 SBPP 中的启动约束 δ.

(3) 在考虑释放时间(delivery time)的多处理器调度问题[1]中(schedule on identical parallel machines),在某些情况下,如果相同类型的任务连续执行,则可以省略释放时间.如果要求在固定时间内完成所有任务并尽可能

* 收稿日期: 2000-01-25; 修改日期: 2000-08-22
基金项目: 国家重点基础研究发展规划 973 资助项目(G1998030403)

作者简介: 顾晓东(1975－),男,江苏江阴人,博士生,主要研究领域为复杂性理论,组合优化;许胤龙(1963－),男,安徽庐江人,副教授,主要研究领域为复杂性理论,并行算法;陈国良(1938－),男,安徽颍上人,教授,博士生导师,主要研究领域为并行与分布式计算,复杂性理论;黄刘生(1956－),男,安徽泰湖人,教授,博士生导师,主要研究领域为复杂性理论.

减少所使用的处理器数目,这就是一个典型的 SBPP 问题.

为了便于分析,我们假定在 SBPP 问题中各类型的物品具有相同的启动放置空间δ.但在实际应用中,各类型的物品可能具有不同的启动放置空间,这时可以根据问题本身的性质通过适当的调整使问题满足 SBPP 的假设(例如,取启动放置空间的最大值等).

经典一维装箱问题是 NP 完全问题[2],求解经典一维装箱问题的近似算法和算法的评测标准已经形成了相当完备的体系[3~5].由于当各输入物品属于同一种类型时,SBPP 退化为经典一维装箱问题,所以 SBPP 也是一个 NP 完全问题.如果 P≠NP,NP 问题不存在有效时间内求得精确解的算法,因此,本文重点考虑 SBPP 问题的各种近似算法,并参考经典一维装箱算法的评测标准来分析这些近似算法.

本文首先给出一个脱线近似算法 C-NF,它首先按照物品所属类型适当调整输入物品的顺序(借助计数排序可以在 $O(n)$时间内完成),然后对这个新的物品序列使用经典的 NF 装箱算法.C-NF 算法是线性算法,其最坏情况渐近性能比是 2,与启动空间的大小无关.本文也考虑了 SBPP 的在线算法,指出大多数在经典一维装箱问题中性能较好的在线装箱策略都不适于用来求解 SBPP 问题:当它们被用来求解 SBPP 问题时都不存在确定的最坏情况渐近性能比.本文还提出了一种在线算法,对任何的启动空间δ,该算法都存在一个非无穷大的最坏情况渐近性能比.

本文第 1 节简单介绍经典一维装箱问题的近似算法和近似算法的评测标准.第 2 节提出 SBPP 问题的脱线算法 C-NF,并分析该算法的性能.第 3 节讨论 SBPP 问题的在线特性.

1 相关知识

经典一维装箱问题是:给定 n 个物品的序列 $L_n=(a_1,a_2,\ldots,a_n)$,物品 $a_i(1\le i\le n)$的大小 $s(a_i)\in(0,1)$,要求将这些物品装入单位容量的箱子 $B_1,B_2,\ldots,B_m$ 中,使得每个箱子中的物品大小之和不超过 1,并使所用箱子的数目 m 最小.

对于给定的经典装箱问题近似算法 A,记 $A(L)$是算法 A 对输入物品序列 L 操作所使用的箱子数目,$\mathrm{OPT}(L)$是最少必须使用的箱子数目.最坏情况渐近性能比和在线特性是评测近似装箱算法性能的重要标准:

定义 **1.** 对于一个近似装箱算法 A,如果存在常数 R 使得对于任何输入物品序列 L,$A(L)\le R\cdot\mathrm{OPT}(L)+c$($c$ 为常数)成立,并且常数 R 不能更小,则称 R 为近似算法 A 的最坏情况渐近性能比,记为 R_A^∞.即

$$R_A^\infty=\inf\{r\le 1:\text{存在 } N>0,\text{当 } \mathrm{OPT}(L)\ge N \text{ 时},A(L)/\mathrm{OPT}(L)\le r \text{ 恒成立}\}. \quad (1)$$

为区别起见,用 R_A^∞ 表示算法 A 求解经典一维装箱问题的最坏情况渐近性能比,而用 R_∞^A 表示算法 A 求解受启动约束装箱问题的最坏情况渐近性能比.同样,若非特别说明,经典装箱算法指的是经典一维装箱问题的近似算法.

定义 **2.** 如果近似装箱算法 A 依序处理各输入物品,在处理当前物品的时候,不知道任何后续物品信息,并立即给出当前物品的装箱方案,则称这样的近似装箱算法 A 为在线(on line)装箱算法;而首先得到所有物品信息之后,统一处理所有物品装箱方案的近似算法则为脱线(off line)装箱算法.在实际应用中,往往要求算法具有在线特性.

算法的时间复杂度和平均性能比是评测近似算法性能的另两个重要标准,表 1 列出了部分著名装箱问题近似算法的研究现状[3].最早提出的在线装箱算法是“下次适应算法”(next fit,简称 NF)[4,6],它按顺序依次处理各物品:首先把物品 a_1 放入箱子 B_1 中;再考虑物品 a_2,如果 a_2 能够放入 B_1,则将 a_2 放入 B_1 中,否则关闭箱子 B_1,打开一个新的箱子 B_2,并将 a_2 放入 B_2 中;然后按照相同的方法依序处理各物品,在处理物品 a_i 时,假设已使用的箱子是 $B_1,B_2,\ldots,B_t$,它只考虑箱子 B_t 中能否放下物品 a_i,而不管 $B_1,B_2,\ldots,B_{t-1}$ 是否有足够的空间放下这个物品.

下次适应算法的性能较差,对它改进后的适应算法在处理物品 a_i 时,如果所有已使用的箱子都不能放下物品 a_i,则打开一个新的箱子并把物品 a_i 放入这个箱子中;否则考虑已使用的箱子 $B_1,B_2,\ldots,B_t$ 中所有能够放下 a_i 的箱子 $B_1',B_2',\ldots,B_s'$,使用一种选择策略从中选出一个箱子并把物品 a_i 放入该箱子中.当使用的选择策略分别为选择第 1 个(下标最小的)、选择最佳的(已装载物品大小和最大的)、选择最坏的(已装载物品大小和最小的)的时候,分别得到经典一维装箱问题的首次适应(first fit,简称 FF)算法、最佳适应(best fit,简称 BF)算法和最坏适应(worst fit,简称 WF)算法.这 3 种算法具有一种共同的特殊性质.

定义 **3.** 对于一个在线近似装箱算法 A,如果在处理当前物品的时候,至少有一个已经打开的物品能够放下这个物品,则把它放入某个能够放下该物品的箱子中而不是打开一个新的箱子.我们称满足这个性质的在线近似装箱算法为任意适应算法(any fit).显然,FF 算法、BF 算法和 WF 都是任意适应算法.

Table 1 Some famous approximation algorithms for classical one-dimensional bin packing
表 **1** 经典一维装箱问题的几种著名近似算法

Approximation algorithms(1)	On line?(2)	Time complexity(3)	Asymptotic worst-case performance ratio(4)	Average-Case performance ratio under (0,1] uniform distribution(5)
Next fit(6)	On line	$O(n)$	2	4/3
First fit(7)	On line	$O(n\log n)$	1.7	1
Best fit(8)	On line	$O(n\log n)$	1.7	1
Worst fit(9)	On line	$O(n\log n)$	2	Not studied(17)
Harmonic-K(10)[6]	On line	$O(n)$	K-Relative(15)	K-Relative
Bounded space algorithms(11)	On line	$O(n)$	Relative to strategy(16)	Relative to strategy
First fit decreasing(12)	Off line(14)	$O(n\log n)$	11/9	1
Best fit decreasing(13)	Off line	$O(n\log n)$	11/9	1

(1)近似算法,(2)是否在线算法?(3)时间复杂度,(4)最坏情况渐近性能比,(5)(0,1)均匀分布下平均性能比,(6)下次适应,(7)首次适应,(8)最佳适应,(9)最坏适应,(10)K-调和,(11)Bounded space 算法类,(12)降序首次适应,(13)降序最佳适应,(14)脱线算法,(15)与 K 相关,(16)具体策略相关,(17)尚未见相关研究结果.

下一节首先尝试设计带启动约束装箱问题的脱线算法并给出算法分析.

2 脱线算法 C-NF

本节首先给出一种求解 SBPP 的脱线算法 C-NF,其最坏情况渐近性能比为 2,与启动空间δ 无关;然后给出了 C-NF 平均性能比的实验分析.

2.1 C-NF算法及其最坏情况渐近性能比分析

受启动约束的装箱问题中,在同一个箱子中放入两个或两个以上同种类物品只要占用一个额外的启动放置空间,因此,把同种类的物品尽可能地放置在相同的箱子中可以减少额外的启动放置空间.一种自然的策略是,首先把同种类的物品排列在一起,然后再对这个新的物品序列使用经典装箱算法.基于此思想,我们提出 C-NF(classified-next fit)算法.

C-NF 算法. C-NF 首先把输入物品序列按照所属类型排序,使具有相同类型的物品聚在一起;然后对这个新的物品序列使用下次适应策略 NF(唯一不同的是,当把物品放入箱子中时,如果箱子中还没有该种类的其他物品,则需额外的启动放置空间δ).

输入物品序列 $L_n=(a_1,a_2,\ldots,a_n)$,令 $L'_n=(a'_1,a'_2,\ldots,a'_n)$是按照类型排序后的新序列,则 C-NF 算法按照 L'_n 中的顺序依次处理各物品,首先把 a'_1 连同δ 启动空间放入箱子 B_1 中;然后考虑 a'_2,如果 a'_2 能够放入箱子 B_1 中,则把 a'_2 放入 B_1 中(如果 a'_2 与 a'_1 是不同类型的,则还要求 B_1 中能另外再放下一个启动空间δ),否则,打开一个新的箱子 B_2,并把 a'_2 连同启动空间δ 放入 B_2 中;按照与 a'_2 相同的方法处理后续物品 $a'_3,\ldots,a'_n$.

C-NF 算法首先对所有输入物品排序,因此它是一个脱线算法.由于物品类型只可以取 1 和 n 之间的整数,则利用"计数排序"把输入物品序列按照类型排序的运行时间为 $O(n)$;使用下次适应算法处理排序后物品序列的运行时间也是 $O(n)$,所以 C-NF 算法的总运行时间是线性的.下面的定理 1 分析了 C-NF 算法的最坏情况渐近性能比.

定理 **1.** 对任何输入物品序列 L,C-NF(L)≤2OPT(L)+1;C-NF 算法的最坏情况渐近性能比是 $R_\infty^{\text{C-NF}}=2$.

证明:设 L 中的物品类型数为 t.C-NF 算法首先对 L 按照物品类型排序,然后把物品放入箱中,设所用的箱子为 $B_1,B_2,\ldots,B_m$,则 C-NF(L)=m.排序后 L 中相同类型的物品聚在一起,考虑各类型的第 1 个物品,设其中有 t_1 个类型的第 1 个物品同时是某个 $B_i(1\le i\le m)$中的第 1 个物品(例如,类型 1 的第 1 个物品同时是 B_1 的第 1 个物品),显然 $t_1\le t$.

令 $s(B)$表示算法结束后箱子 B 被占用的空间,它分成两部分:一部分是 B 中物品占据的空间,记为 $s_1(B)$;另一

部分用于启动空间,记为 $s_2(B)$,则 $s(B)=s_1(B)+s_2(B)$.并令 $s_1(L)=s_1(B_1)+s_1(B_2)+\ldots+s_1(B_m)$,$s_2(L)=s_2(B_1)+s_2(B_2)+\ldots+s_2(B_m)$.

考虑 $s_1(B_i)+s_1(B_{i+1})$,$1\le i<m$:① 若 B_{i+1} 中的第 1 个物品与 B_i 中最后一个物品是相同类型的,根据下次适应算法的定义,B_{i+1} 中第 1 个物品 a 不能放在 B_i 中,则 $s(B_i)+s(a)>1$,所以 $s_1(B_i)+s_1(B_{i+1})\ge s_1(B_i)+s(a)>1-s_2(B_i)$;② 否则,类似于情况①的分析可得 $s_1(B_i)+s_1(B_{i+1})>1-s_2(B_i)-\delta$.

在情况②中,B_{i+1} 中的第 1 个物品与 B_i 中最后一个物品的类型不同,即 B_{i+1} 中第 1 个物品是某种类型 $k(k>1)$ 的第 1 个物品,因此共有 t_1-1 个 $s_1(B_i)+s_1(B_{i+1})$ 满足情况②.将 $s_1(B_i)+s_1(B_{i+1})$ 对所有 $1\le i<m$ 求和,得

$$2s_1(L)>m-1-s_2(L)-(t_1-1)\delta .$$

由于 $s_2(L)$ 由若干启动空间组成,令 $s_2(L)=p\delta$,则上式等价于

$$\text{C-NF}(L)<2s_1(L)+p\delta+(t_1-1)\delta+1. \tag{2}$$

设最佳装箱策略 OPT(L)中用了 q 个启动放置空间,则

$$\text{OPT}(L)\ge s_1(L)+q\delta. \tag{3}$$

下面来考虑 p 和 q 的关系.令 L 中物品类型为 1,2,…,t,并设 C-NF 算法中类型为 i 的物品共使用了 p_i 个启动空间,而在 OPT 算法中则仅使用 q_i 个启动空间,则 p_i,q_i 分别为 C-NF 算法、OPT 算法中类型为 i 的物品所占用的箱子数目.

对经典一维装箱问题的任意物品序列 L'',有[4]

$$\text{NF}(L'')\le 2\text{OPT}(L'')-1. \tag{4}$$

而当第 i 类型的第 1 个物品是它所在箱子的第 1 个物品时(共有 t_1 个类型满足该条件),对该类型物品的处理等价于 NF 操作 BPP,由不等式(4)可得 $p_i\le 2q_i-1$;否则,该类型物品所占用的第 1 个箱子是与其他类物品共享的,类似前一种情况可得 $p_i\le 2q_i$.把 p_i 对所有 $1\le i\le m$ 求和,可得(因为 p, q 分别为 p_i, q_i 对 i 的求和)

$$p\le 2q-t_1< 2q-(t_1-1). \tag{5}$$

联合式(2)、式(3)和式(5),得

$$\text{C-NF}(L)< 2\text{OPT}(L)+1. \tag{6}$$

用(a,c)表示大小为 a、类型为 c 的物品,$\{a_1,a_2,\ldots,a_n\}_k$ 表示把物品序列$\{a_1,a_2,\ldots,a_n\}$重复 k 次所得的物品序列.考虑输入物品序列:

$$L=\{(\frac{1-\delta}{2},1),(\frac{1-\delta}{2N},1)\}_{2N}.$$

L 中所有物品属于同一个类型,使用 C-NF 算法时每两个物品占用一个箱子而后一个物品恰好不能放入这个箱中,所以 C-NF(L)=2N;而若使用最佳策略,所有大小为$(1-\delta)/2N$ 的物品恰好可以放入一个箱子中,而每两个大小为$(1-\delta)/2$ 的物品也恰好占用一个箱子,故 OPT(L)=N+1.即存在任意长输入序列满足 C-NF(L)=2OPT(L)–2.

综上所述,C-NF 算法的最坏情况渐近性能比为 2. □

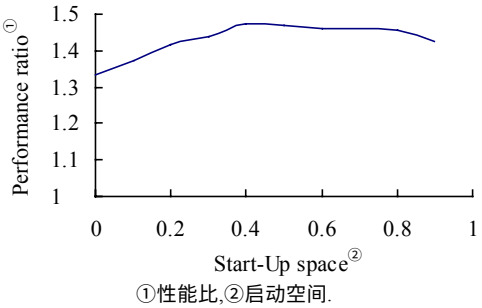
根据定理 1,不管δ 取何值,C-NF 算法的最坏情况渐近性能比都是 2.如果把 C-NF 算法的思想看作一种策略,即先把输入物品序列按照物品类型排序,然后对排序后的新序列使用经典装箱策略,如 FF,BF,WF,则可以类似地定义 C-FF 算法、C-BF 算法和 C-WF 算法.它们由于先进行预排序,所以都是脱线算法.用与定理 1 完全相同的证明,我们可以得到 C-WF 算法的最坏情况渐近性能比.

推论 **1.** C-WF 算法求解 SBPP 的最坏情况渐近性能比是 $R_\infty^{\text{C-WF}}=2$.

2.2 平均性能比的实验分析

图 1 给出了 C-NF 算法的平均性能比与启动空间δ 之间关系的实验分析,其中算法的平均性能比是 C-NF 算法对 100 个输入规模为 15 360 的实例的性能比取平均值,所取的实例是依照 OR-Library* 构造经典一维装箱问题实例的方法[7]构造的.

* http://mscmga.ms.ic.ac.uk/info.html



①性能比,②启动空间.

Fig.1 Relationship between the performance ratio and start-up space of algorithm C-NF

图 1 C-NF 算法的性能比与启动空间的关系

在用实验分析装箱算法的性能比时,一个最关键的问题就是对最佳装箱方案使用的箱子数 OPT(L)的估计.由于装箱问题(包括 BPP 和 SBPP)是 NP 完全问题,故当前不存在有效时间内求得大规模实例的最佳解的方法.我们在实验中取 OPT(L)的估计值为

$$\max\{s(L)/(1-\delta), s(L)+u\delta\} \quad (\text{其中 } u \text{ 是实例中的物品类型数}). \tag{7}$$

在最佳方案中 $\mathrm{OPT}(L)\geq s(L)+\mathrm{OPT}(L)\delta$ (因为每个箱子中至少有一个启动空间),故 $\mathrm{OPT}(L)\geq s(L)/(1-\delta)$;又因为 $\mathrm{OPT(L))}\geq s(L)+u\delta$,可知使用式(7)估计 OPT($L$)值将造成系统误差,使 OPT($L$)值略偏小,所得算法的平均性能比要比实际结果偏大.但是当输入物品序列 L 的规模足够大时,式(7)是 OPT(L)一个很好的估计.

3 SBPP 问题的在线特性

本节指出了用在线算法求解 SBPP 问题的困难性,特别是当把一些常见在线装箱策略(包括表 1 中所有的在线算法和所有其他的任意适应算法)应用在 SBPP 问题上时,不存在确定的最坏情况渐近性能比.在下一节给出了一种求解 SBPP 问题的在线算法,它对于任何δ 都具有确定的最坏情况渐近性能比.

3.1 经典在线装箱算法

在带启动约束的装箱问题 SBPP 中处理某种新类型物品时,如果不知道后续物品的信息,很难抉择是把它放入某个非空箱子中以提高当前利用率,还是放入一个新打开的箱子中以保留剩余空间给其他同种类物品.因此,用在线算法求解 SBPP 有很高的难度.

在经典一维装箱问题中,任意适应算法是一类具有极佳的时间复杂度、最坏情况渐近性能比、平均情况渐近性能比的在线装箱算法.这一类算法在处理当前物品的时候,如果至少存在一个已经打开的箱子能够放下这个物品,则不打开一个新的箱子,这也是一种很常见的贪心策略.然而,在带启动约束的装箱问题中,任何任意适应算法都不具有一个确定的最坏情况渐近性能比.

定理 2. 任意适应算法求解 SBPP 问题不存在确定的最坏情况渐近性能比.令 A 是求解 SBPP 问题的一个近似算法.如果 A 属于任意适应算法,则对任何整常数 $N_1,u\geq 2$,都存在接近 $1/u$ 的启动空间δ 和任意长输入序列 L,使得 $A(L)/\mathrm{OPT}(L)>N_1$.

证明:用(a,c)表示大小为 a、类型为 c 的物品,常数 $N\in Z^+$,启动空间δ.

令 $k=\left\lfloor \dfrac{1}{(1+1/N)\delta} \right\rfloor$, $t=\left\lfloor \dfrac{1-k(1+1/N)\delta}{\delta/N} \right\rfloor=\left\lfloor \dfrac{N}{\delta} \right\rfloor \bmod(N+1)$,以及 $\varphi=\delta/N$.

考虑任意长物品序列

$$L=\{(\varphi,1),(\varphi,2),...,(\varphi,k),\{(\varphi,1)\}_t\}_{(kN+k-N+t)s} \quad (\text{其中 } s\in Z^+).$$

其输入规模 $n=(k+t)(kN+k-N+t)s$,图 2 表示了用任意适应策略和最佳策略对序列 L 操作的结果.当使用任意适应算法 A 时,由于 L 中前面$(k+t)$个物品能够放在同一个箱子中,根据定义 3,任意适应算法 A 把这$(k+t)$个物品放入同一个箱子中,而此时箱子的剩余容量小于 φ ,故不能再放下任何一个后续物品,因此所有物品按照恰好每$(k+t)$个物品占用一个箱子放置,所以 $A(L)=(kN+k-N+t)s$;而在最佳策略中,把同类型的物品放在同一个箱子中,每个箱

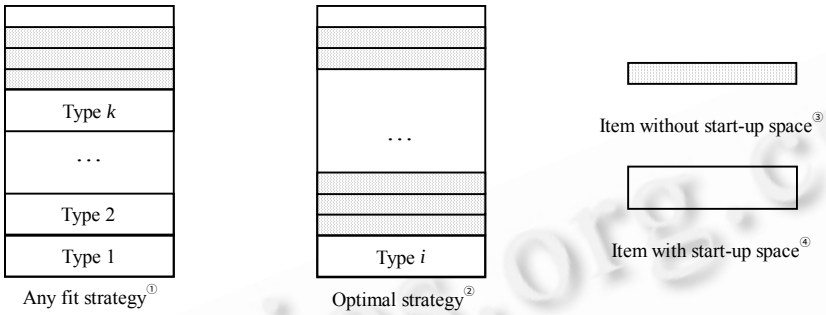
子只需要一个启动放置空间,故可以放置 $kN+k-N+t$ 个物品,所以 $\mathrm{OPT}(L)=(k+t)s$.因而

$$A(L)/\mathrm{OPT}(L)=(kN+k-N+t)/(k+t)=1+N(k-1)/(k+t). \tag{8}$$

取 $N=3N_1,\delta=1/(u+(u+1)/N)$,则 $t=1$;又因为 $u\geq 2$,所以$\delta<1/2$,根据定义得 $k>1$.由式(8)可得

$$A(L)/\mathrm{OPT}(L)>1+N/3>N_1.$$

故定理 2 成立. □



①任意适应策略,②最佳策略,③不需要启动空间的物品,④需要启动空间的物品.

Fig.2 Packing of input item list L

图 2 输入序列 L 的放置策略

根据定理 2,如果要给出一个解 SBPP 的具有确定最坏情况渐近性能比的在线算法,则这个算法必须摒弃任意适应这个特性.定理 3 进一步说明,表 1 中的任何一种在线策略都不适合用来解 SBPP 问题.

定理 **3.** 用表 1 中的任何一个在线算法 A 来求解 SBPP 问题都不存在确定的最坏情况渐近性能比.

证明:① 因为首次适应算法 FF、最佳适应算法 BF 和最坏适应算法 WF 都属于任意适应算法,由定理 2,它们应用在 SBPP 上都不存在确定的最坏情况渐近性能比.

② 对于下次适应算法 NF,令 $t=\lceil 1/\delta\rceil-1$,$\varepsilon$ 是一个无穷小量,取输入物品序列:

$$L=\{(\varepsilon,1),(\varepsilon,2),...,(\varepsilon,2t)\}_m.$$

若使用 NF 策略,则每 t 个物品放在一个箱子中,每个物品要附带启动放置空间,使用的箱子数 $\mathrm{NF}(L)=2m$;而最佳放置策略把所有同类型的物品放在同一个箱子中(因为物品大小均为无穷小量 ε),所以 $\mathrm{OPT}(L)=2t$.由于 t 是一个依赖于δ 的常数,所以只要选择 m 足够大,$R_\infty^{\mathrm{NF}}=+\infty$.故 NF 策略解 SBPP 问题不存在确定的最坏情况渐近性能比.

③ 调和算法 $\mathrm{H_K}$ 把物品按照大小分类,而在同一类内部使用 NF 策略[8].由于②中给出的输入实例 L 中所有物品大小相等,故调和算法把它们分在同一类中.因此对 L,$\mathrm{H_K}$ 与 NF 给出相同的装箱方案,故 $\mathrm{H_K}$ 解 SBPP 问题不存在确定的最坏情况渐近性能比.

④ 所有的 Bounded Space 算法在选择的参数 $K\to\infty$时就是 NF,FF 和 BF 三者之一,所以这类算法解 SBPP 问题也不存在确定的最坏情况渐近性能比.

综合上述,定理 3 成立. □

由定理 2 和定理 3,设计带启动约束装箱问题的在线算法具有其固有的复杂性.在下一节,本文给出一种求解 SBPP 问题的具有确定最坏情况渐近性能比的在线算法.

3.2 在线算法FC-NF

基于与 C-NF 类似的思想,通过将箱子进行分类,我们提出了 SBPP 的具有非无穷大、与δ 相关的最坏情况渐近性能比的在线算法 FC-NF(fully classified-next fit).

FC-NF 算法. FC-NF 算法首先把箱子分类:类型为 i 的箱子中只放类型为 i 的物品(因此每个箱子中只需要一个启动放置空间).而对于所有相同类型的箱子,其中的物品放置采用经典装箱策略 NF.

FC-NF 算法是线性在线算法,而不是任意适应算法.定理 4 表明对任意δ,FC-NF 算法具有非无穷大的最坏情况渐近性能比.

定理 **4.** 线性在线算法 FC-NF 的最坏情况渐近性能比满足:① 当$\delta\le 1/3$ 时,$\lceil 1/\delta\rceil-1\le R_\infty^{\text{FC-NF}}\le 1/\delta$;② 当$1/3<\delta<1/2$ 时,$2\le R_\infty^{\text{FC-NF}}\le 2/(1-\delta)$;③ 当$\delta\ge 1/2$ 时,$R_\infty^{\text{FC-NF}}=2$.

证明:令(a,c)表示大小为 a,类型为 c 的物品.首先构造如下任意长输入物品序列 $L=\{(\varepsilon,1),(\varepsilon,2),\ldots,(\varepsilon,n)\}$,其中$\varepsilon$是一个无穷小量.使用 FC-NF 算法,每一个物品占用一个箱子,所以 FC-NF(L)=n;而对于最佳策略,有 $\text{OPT}(L)=n/(\lceil 1/\delta\rceil-1)$,所以,对任意的 A 和δ,

$$R_\infty^{\text{FC-NF}}\ge\lceil 1/\delta\rceil-1. \tag{9}$$

设输入物品序列 L 中输入物品的类型有 1,2,…,u,FC-NF 算法使用的箱子数目为 m.

设所有类型为 i 的箱子为 $B_1,B_2,\ldots,B_{v(i)}$,则考虑其中任何两个相邻的箱子 B_t和 $B_{t+1}(1\le t<v(i))$.FC-NF 算法在处理这些箱子中的物品时使用经典装箱算法 NF,每个箱子中恰好使用一个启动空间,因为 B_{t+1} 中的第一个物品 a 不放在 B_t 中而是打开一个新的箱子 B_{t+1} 并把 a 放入其中,所以

$$s(B_t)+s(B_{t+1})>1-\delta. \tag{10}$$

对所有 $1\le t<v(i)$,把不等式(10)相加,则

$$2[s(B_1)+s(B_2)+\ldots+s(B_{v(i)})]>(1-\delta)(v(i)-1). \tag{11}$$

对所有箱子类型 $1\le t\le u$,把不等式(11)相加,可得 $2s(L)>(1-\delta)(m-u)$,即

$$\text{FC-NF}(L)=m<2s(L)/(1-\delta)+u. \tag{12}$$

而在最佳装箱方案中,每种类型的物品至少使用一个启动空间,所以

$$\text{OPT}(L)\ge s(L)+u\delta. \tag{13}$$

把式(12)、式(13)两式相除得 $\text{FC-NF}(L)/\text{OPT}(L)\le[2s(L)/(1-\delta)+u]/(s(L)+u\delta)$,故

$$R_\infty^{\text{FC-NF}}=\text{FC-NF}(L)/\text{OPT}(L)\le\max\{[2s(L)/(11-\delta)]/s(L),u/(u\delta)\}. \tag{14}$$

① 当$\delta\le 1/3$ 时,$2/(1-\delta)\le 1/\delta$,联合式(9)、式(14)可得$\lceil 1/\delta\rceil-1\le R_\infty^{\text{FC-NF}}\le 1/\delta$.

② 当 $1/3<\delta<1/2$ 时,$2/(1-\delta)>1/\delta$,联合式(9)、式(14)得 $2\le R_\infty^{\text{FC-NF}}\le 2/(1-\delta)$.

③ 当$\delta\ge 1/2$ 时,不管使用什么装箱策略(包括最佳策略),每一个箱子中都只能放入一种类型的物品.所以,FC-NF 算法解 SBPP 问题与算法 NF 解经典 BPP 问题 u 次等价,由式(4)可得 $\text{FC-NF}(L)\le 2\text{OPT}(L)-u$.所以 $R_\infty^{\text{FC-NF}}\le 2$,而当 $u=1$ 时问题与经典 BPP 问题等价,所以 $R_\infty^{\text{FC-NF}}\ge R_{\text{NF}}^{\infty}=2$.所以, $R_\infty^{\text{FC-NF}}=2$.

综合上述,定理 4 成立. □

4 小 结

本文提出了一种带约束的装箱问题 SBPP.它考虑待装箱物品的内在关联,在经典装箱问题的各种应用中,实际上都存在着这种内在关联的约束.本文首先给出了一种求解 SBPP 的脱线算法 C-NF,其最坏情况渐近性能比 2 与启动空间大小无关;然后分析了问题本身的在线特性,指出大量的经典在线装箱策略都不适于用来求解 SBPP 问题;最后给出了一种对任何启动空间δ,都存在确定的最坏情况渐近性能比的在线装箱策略 FC-NF.

设计 SBPP 的在线装箱算法使其具有与启动空间δ 无关的最坏情况渐近性能比,以及设计最坏情况渐近性能比小于 2 的脱线装箱算法,是我们当前的研究课题.

References:

[1] Hall, L.A. Approximation algorithms for scheduling. In: Hochbaum, D., ed. Approximation Algorithms for NP-Hard Problems. Boston: PWS Publishing, 1996. 1~45.

[2] Garey, M.R., Johnson, D.S. Computers and Intractability: a Guild to the Theory of NP-Completeness. New York: W.H. Freeman and Company, 1978.

[3] Coffman, E.G., Garey, M.R., Johnson, D.S. Approximation algorithms for bin packing: a survey. In: Hochbaum, D., ed. Approximation Algorithms for NP-Hard Problems. Boston: PWS Publishing, 1996. 46~93.

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

[5] van Vliet, A. On the asymptotic worst-case behavior of harmonic fit. Journal of Algorithms, 1996,20(1):113~136.

[6] Coffman, E.G., So, K., Hofri, M., *et al*. A stochastic model of bin-packing. Information and Control, 1980,44(2):105~115.

[7] Falkenauer, E. A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics, 1996,2(2):5~30.

[8] Lee, C.C., Lee, D.T. A Simple on-line bin packing algorithm. Journal of ACM, 1985,32(3):562~572.

On a Constrained Bin Packing Problem with Start-Up Space*

GU Xiao-dong, XU Yin-long, CHEN Guo-liang, HUANG Liu-sheng

(*National High Performance Computing Center at Hefei*, *Hefei* 230027, *China*);

(*Department of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei* 230027, *China*)

E-mail: gxd@mail.ustc.edu.cn

http://www.nhpcc.ustc.edu.cn

Abstract: A constrained bin packing problem with start-up space (SBPP) is proposed in this paper, in which an additional start-up space is needed if different items are put into a same bin. The problem has many applications such as job allocation, multiprocessor scheduling and real-world packing. A linear offline approximation algorithm C-NF is presented to solve the SBPP problem. It is proved that the C-NF algorithm has an asymptotic worst-case performance ratio of 2, which is independent of the size of start-up space. And the experimental average-case performances of C-NF are given. Also, the online property of SBPP is studied. It is pointed out that most of the classic online algorithms cannot offer definite worst-case performance ratios when applied on SBPP. And an online algorithm is proposed with a finite asymptotic worst-case performance ratio for any start-up space.

Key words: bin packing problem; combinatorial optimization; approximation algorithm; asymptotic worst-case performance ratio; average-case performance ratio

* Received January 25, 2000; accepted August 22, 2000
Supported by the National Grand Fundamental Research 973 Program of China under Grant No.G1998030403