求解三维装箱问题的启发式正交二叉树搜索算法

刘 胜[1),2)] 朱凤华[1),2)] 吕宜生[1)] 李元涛[1)]

[1)](中国科学院自动化研究所复杂系统管理与控制国家重点实验室 北京 100190)

[2)](中国科学院云计算产业技术创新与育成中心 广东 东莞 523808)

摘 要 文中提出了一种求解三维装箱问题的启发式二叉树搜索算法,首先将所有箱子组合成多个优条,每个优条中的箱子沿容器高度方向排成一列;接着开始构建二叉树,其根节点表示空的装箱方案,每个树节点沿长度方向增加一排优条形成左子树节点,沿宽度方向增加一排优条形成右子树节点,二叉树必须扩展到所有叶子节点都无法再放入任何剩余的箱子为止,所有叶子节点中填充率最高的装箱方案即为最终结果.该算法满足三维装箱的 3 个著名的约束条件.在多样性最强的测试算例中,该文方法相对于现有最优秀装箱算法装箱率有显著提高.

关键词 三维装箱;启发式算法;二叉树

中图法分类号 TP301 **DOI 号** 10.11897/SP.J.1016.2015.01530

A Heuristic Orthogonal Binary Tree Search Algorithm for Three Dimensional Container Loading Problem

LIU Sheng[1),2)] ZHU Feng-Hua[1),2)] LV Yi-Sheng[1)] LI Yuan-Tao[1)]

[1)](*The State Key Laboratory of Management of Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing* 100190)

[2)](*Cloud Computing Center, Chinese Academy of Sciences, Dongguan, Guangdong* 523808)

Abstract This paper presents a heuristic orthogonal binary tree search algorithm for three dimensional container loading problem. Firstly all boxes are composed into multiple good-strips. The boxes in each good-strip are arranged along a vertical line. Then a binary tree is created. Each tree node in the tree is a container-loading plan while the root node denotes an empty one. The left child node of each node is obtained by inserting a line of good-strips along the length of the container into the residual space of the container. Similarly, the right one is obtained by inserting a line of good-strips along the width of the container. The binary tree must be extended so that each leaf tree node must not accommodate any remaining boxes. The result is the container-loading plan with highest fill rate among all the leaf tree nodes. The algorithm satisfies three famous constraints on three dimensional container loading. It outperforms the best known algorithm in the most strongly heterogeneous test data.

Keywords three-dimensional container loading; heuristic algorithm; binary tree

1 引 言

三维装箱问题是物流领域和木材加工领域的经典问题,研究如何提高货箱空间利用率、如何提高木材的利用率等,从而降低用户成本,提高用户利润,具有非常积极的现实意义.

三维装箱问题是切割和布局问题的子问题,旨

收稿日期:2013-05-20;最终修改稿收到日期:2014-12-12.本课题得到国家自然科学基金(61104054,71232006,61233001)资助.**刘 胜**,男,1978 年生,博士,副研究员,主要研究方向为组合优化、智能物流、智能交通、服务计算.E-mail: sheng.liu@ia.ac.cn. **朱凤华**,男,1976 年生,博士,副研究员,主要研究方向为智能交通、智能物流. **吕宜生**,男,1983 年生,博士,助理研究员,主要研究方向为智能交通、组合优化. **李元涛**,男,1979 年生,博士,高级工程师,主要研究方向为智能交通、组合优化.

在研究如何将若干个货物装填到一个或多个大的容器中，在 Dyckhoff 和 Finke[1] 和 Wäscher 等人[2] 的关于切割和布局问题的综述中有详细的论述. 三维装箱问题按照装载的货物形状分类，可分为规则形状(通常为长方体)货物装箱和不规则形状货物装箱；按照货物种类分类，可以分为单一种类货物装箱和多种类货物装箱；按照目标函数分类可以分为容器装载问题、箱柜装载问题和背包装载问题. 这 3 种都属于三维装箱问题的子问题，它们之间的区别由表 1 列出.

表 1　3 种装箱问题的区别

问题	容器数量	容器三维尺寸	货物是否必须全部装入容器	目标函数
容器装载	单个	决策变量	是	容器体积最小
箱柜装载	决策变量	常量	是	容器数量最少
背包装载	单个	常量	否	装入货物体积最大

下面分别介绍以上 3 种类型的三维装箱问题.

(1)容器装载问题(three-Dimensional Container-Packing Problems，3D-CPP). 所有箱子要装入一个不限尺寸的容器中，目标是要找一个装填方案，使得容器体积最小. 该问题更一般的形式是容器的长和宽不变，找一个装填使容器高最小，Bischoff 等人[3] 比较了该问题的不同算法. 陆一平等人[4] 给出了三维矩形块布局的序列三元组编码方法；Bortfeldt 等人[5] 设计了容器装载问题的启发式算法，解决如何在容器宽高不变的前提下，使容器长度尽可能短的问题.

(2)箱柜装载问题(three-Dimensional bin Packing Problem，3D-BPP). 给定一些不同类型的方型箱子和一些规格统一的方型容器，问题是要把所有箱子装入最少数量的容器中. Faroe 等人[6] 提出了该问题的一种启发式算法.

(3)背包装载问题(three-Dimensional Knapsack Loading Problems，3D-KLP，又称为 three-Dimensional Container Loading Problems，3D-CLP). 每个箱子有一定的价值，背包装载是选择箱子的一部分装入容器中，使得装入容器中的箱子总价值最大. 如果把箱子的体积作为价值，则目标转化为使容器浪费的体积最小. 当箱子的长、宽分别与容器的长、宽相等时，问题等价于经典的一维背包问题. 三维装箱问题是典型的 NP 难题，不存在多项式时间复杂度的最优求解算法. 用传统的精确算法求解这类问题，会发生“组合爆炸”的现象，只适合解决箱子种类数量很小的装箱问题，对于实际应用中规模比较大的数据，很难求得它的最优解. 启发式求解方法依然是解决 3D-KLP(3D-CLP)的首选. 因此很多学者提出了针对该问题的启发式算法. Pisinger[7] 将这些算法分为砌墙算法、建堆算法、立方体排列算法和切割算法四大类. George 等人[8] 首次对该问题提出了砌墙算法，该算法把容器分成层，层再分成条，最终转化为一维背包问题；Bischoff 等人[3] 比较了 14 种基于层的方法；Bischoff 等人[9] 提出了按层布局的贪婪算法，针对层和条的选择策略；Pisinger[7] 提出了基于砌墙策略的树搜索算法；Gehring 等人[10] 将箱子生成“塔”，利用遗传算法将“塔”优化组合；Bortfeldt 等人[11] 针对箱子种类数较少的情况提出了一种禁忌搜索算法；Bortfeldt 等人[12] 基于层的概念，提出了一种解决三维装箱问题的混合遗传算法；Eley[13] 设计了一种基于同类块(Block)的算法，基本思想是先将同类的箱子组合成长方体块再装箱；Bortfeldt 等人[14] 进一步拓展“块”的概念，并使用并行禁忌搜索寻找最优的装载方案；Moura 等人[15] 基于“剩余空间”的概念提出了一个贪心随机自适应搜索算法(GRASP)；Parreno 等人[16] 进一步发展和改进了该算法，装箱率有较大提高；Parreno 等人[17] 基于最大空间的概念，提出了一个可变邻域搜索算法；Bortfeldt 等人[5] 设计了一种三维条装载的启发式算法；Fanslau 等人[18] 拓展了“块”的概念，“块”可以包含多种尺寸的箱子，也可以包含同种箱子的多个摆放方向，称之为“复合块”，并设计了一个有效的启发式树状搜索算法；此外 Ngoi 等人[19]、Morabito 等人[20]、Sixt 等人[21]、Gehring 等人[10,22]、Lim 等人[23]、Juraitis 等人[24] 也报告了其他一些针对三维装箱问题的启发式算法. 国内学者对三维装箱问题的研究，取得了很好的结果，何大勇等人[25] 设计了求解复杂集装箱装载问题的遗传算法；张德富等人[26] 提出了一个混合模拟退火算法；Huang 等人[27] 提出了一个有效的拟人型穴度算法；值得一提的是张德富等人[28] 设计的求解三维装箱问题的多层启发式搜索算法，是目前为止有公开文献报导的解决装箱问题的最优秀算法；还有其他一些学者也取得了不错的研究成果[26-31].

实际装箱过程中，通常会遇到以下 3 类约束[18]：

(C1)方向性约束(Orientation Constraint). 在很多场合下，箱子的装载有方向性约束. 并不是箱子的所有边都可以竖直放置，方向性约束定义箱子的长度边、宽度边、高度边是否可以竖直放置，箱子至少有一个边可以竖直放置.

(C2)稳定性约束(Support Constraint).在一些场合如集装箱货运,要求箱子必须满足稳定性约束.每个被装载的箱子必须得到容器底部或者其他已经装载箱子的支撑.稳定性约束包括完全支撑约束和部分支撑约束,完全支撑约束要求被装载箱子的底部必须跟容器底部或其他箱子顶部完全接触,部分支撑约束允许箱子底部可以允许部分悬空.

(C3)完全切割约束(Guillotine Cutting Constraint).如在木材切割的应用中,很多切料机只能做完全切割,即每一刀必须从一边切到另一边,把木材完全分割成两半;还有在依靠叉车装卸货的集装箱装卸应用中,要求叉车每次只从集装箱的底部叉装货物,就可以实现所有货物的装卸,从而大大提高装卸速度.这就要求最终的箱子装填块(又叫装填方案)可以通过多个竖直的平面分割成多个竖层,每一个竖层可以通过多个水平的平面分割为最终的箱子,每次的竖直平面分割(和水平平面分割)必须可以将对应的子装填块完全分割成两个独立的子块.所有的竖直平面和水平平面必须不能与对应装填块中的箱子相交.

目前的算法很多只能满足 C1 和 C2 约束,并且在箱子种类较多时,装箱率不高,还有提高的余地.本文提出了一种三维装箱问题的启发式树搜索算法,能够同时满足 C1、C2 和 C3 约束,并且当箱子种类数量较多时,本文方法相对于现有最优秀装箱算法装箱率有显著提高.本文算法简称为启发式正交二叉树搜索算法(Heuristic Orthogonal Binary Tree Search Algorithm,HOBTS).

2 问题介绍

首先介绍一下 3D-KLP(3D-CLP)问题的形式化定义:给定一个长方型容器 C 和一个长方型箱子的集合 $B=\{b_1,b_2,\cdots,b_n\}$,容器 C 长 l_C、宽 w_C、高 h_C,每个箱子 b_i 有长 l_i、宽 w_i 和高 h_i,每个箱子的体积为 $v_i=l_iw_ih_i$.令 0-1 标志 zh_i、zl_i、zw_i 表示箱子是否允许正放、仰放、侧放(0 表示不允许,1 表示允许),令

$$C=(l_C,w_C,h_C) \tag{1}$$

$$B=\{b_1,b_2,\cdots,b_n\} \tag{2}$$

$$b_i=(l_i,w_i,h_i,zh_i,zl_i,zw_i) \tag{3}$$

图 1 为容器尺寸及坐标系示意图;图 2 为箱子的尺寸和摆放方向示意图.

设 F 为 B 的一个子集,定义 F 中所有箱子体积之和为 V_F,即

$$V_F=\sum_{b_i\in F}v_i.$$

问题的目标是选择 B 的一个子集使得 V_F 最大,并且满足以下条件:(1)对任何箱子 $b_i\in F$,在容器 C 中对应一个装填位置;(2)所有 F 中的箱子必须全部包含在 C 中;(3)F 中任何两个箱子在容器中放置后都不重叠;(4)F 中箱子 b_i 的方向必须满足方向标志 zh_i、zl_i、zw_i 要求.定义子集 F 对应的填充率为 $FR_F=V_F/(l_Cw_Ch_C)$.

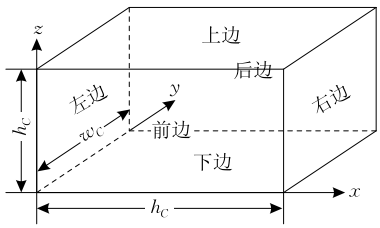

图 1 容器尺寸及坐标系

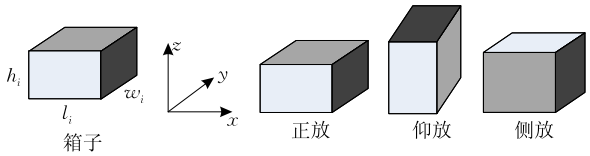

图 2 箱子尺寸及摆放方向

带有完全支撑约束的三维装箱问题,要求所有箱子的底部都完全和容器底部或者其他箱子的顶部完全接触,并且装入容器的箱子体积之和尽可能大.

3 三维装箱算法

在介绍本文算法之前,先进行如下定义:

优条 gs:针对长方型容器 C 和长方型箱子的集合 B,从 B 中选择一个子集 Q,Q 中的箱子在满足方向标志 zh_i、zl_i、zw_i 要求的前提下,沿 z 轴竖直地摞起一列,形成竖条 s,竖条 s 在 z 轴方向尺寸不大于容器高度 h_C,分别定义 l_s、w_s 为竖条 s 中箱子在 x 轴方向最大尺寸和 y 轴方向最大尺寸.

定义竖条 s_j 填充率:

$$fr\_s_j=V_Q/(l\_s_j\times w\_s_j\times h_C) \tag{4}$$

优条即为 B 中 fr_s_j 值最大的子集 Q 形成的竖条,以 gs 表示优条,令

$$gs=(Q,l\_s,w\_s) \tag{5}$$

优条集合 GS:多个优条构成的集合,设 GS 中包含 n 个优条,则有

$$GS=\{gs_1,gs_2,\cdots,gs_n\} \tag{6}$$

优层 gl:针对长方型容器 C 和优条集合 $GS=$

$\{gs_1, gs_2, \cdots, gs_m\}$，从 GS 中选择一个子集 T，T 中的优条沿 x 轴或 y 轴方向排成一行（排列过程中优条可以绕 z 轴旋转 90°），形成竖层 l，其中沿 y 轴方向排列时，沿 y 轴方向的尺寸不大于给定尺寸 w_left（该值对应于生成优层前集装箱内部剩余空间的宽度）；沿 x 轴方向排列时，沿 x 轴方向的尺寸不大于给定尺寸 l_left（该值对应于生成优层前集装箱内部剩余空间的长度）.

定义 l_l 为沿 y 轴方向排列时 T 中优条在 x 轴方向占用最大尺寸.

定义 w_l 为沿 x 轴方向排列时 T 中优条在 y 轴方向占用最大尺寸.

以定义层 l_j 填充率：

$$fr\_l_j = \begin{cases} \sum_{gs_i \in T} V_{gs_i.Q} / (l\_left_j \times w\_l_j \times h_C), \\ \quad \text{当竖层没 } x \text{ 方向摆放时；} \\ \sum_{gs_i \in T} V_{gs_i.Q} / (w\_left_j \times l\_l_j \times h_C), \\ \quad \text{当竖层沿 } y \text{ 方向摆放时.} \end{cases} \tag{7}$$

设 l_j 为优条沿 x 轴方向排放时，GS 中 fr_l_j 值最大的子集 T_j 形成的竖层，设 l_k 为优条沿 y 轴方向排放时，GS 中 fr_l_k 值最大的子集 T_k 形成的竖层，则有

$$gl = \begin{cases} l_j, & \text{如果}(fr\_l_j \geq fr\_l_k) \\ l_k, & \text{否则} \end{cases} \tag{8}$$

令

$$gl = (T, dir, length, thick) \tag{9}$$

其中 dir 含义如下

$$dir = \begin{cases} 0, & \begin{cases} \text{如果 } gl \text{ 沿 } x \text{ 轴方向摆放，} \\ thick \text{ 表示 } w\_l, \\ length \text{ 表示 } x\_left \text{（容器剩余空间长度）} \end{cases} \\ 1, & \begin{cases} \text{如果 } gl \text{ 沿 } y \text{ 轴方向摆放，} \\ thick \text{ 表示 } l\_l, \\ length \text{ 表示 } y\_left \text{（容器剩余空间宽度）} \end{cases} \end{cases} \tag{10}$$

优层集合 GL：多个优层构成的集合，设 GL 中包含 m 个优层，则有

$$GL = \{gl_1, gl_2, \cdots, gl_m\} \tag{11}$$

本算法先将所有箱子组合成多个沿容器 z 方向摆放的优条，将三维装箱问题降维为以优条为装填对象的二维装箱问题，然后构造二叉树，如图 3 所示，树上的每个节点对应一个填充方案，根节点对应空的填充方案，在每个节点对应的填充方案基础上沿 x 方向摆放一排优条，形成新的填充方案作为其左子树节点对应的填充方案，在每个节点的填充方案基础上沿 y 方向摆放一排优条，形成新的填充方案作为其右子树节点对应的填充方案，叶子节点对应的是无法再填充剩余箱子的填充方案，最后以填充率最高的叶子节点的装箱方案最为最终装箱方案.

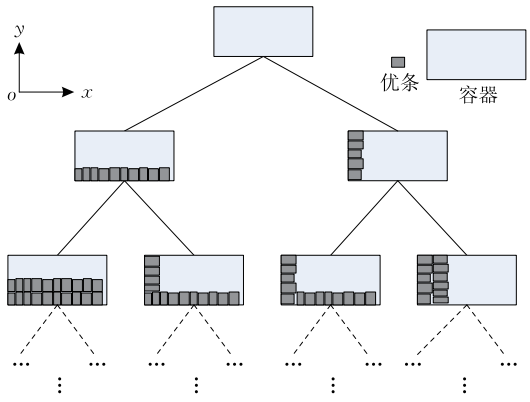

图 3　三维装箱方案搜索二叉树

三维装箱算法的实现流程如算法 1 所示. 首先生成空的优层集合 GL，定义最优的优层集合 $BEST\_GL$，为其赋初始值 GL；接着定义 x 方向剩余尺寸 x_left，初始值为 l_C，y 方向剩余尺寸 y_left，初始值为 w_C；再生成空的优条记录集合 GS_REC 和空的优层记录集合 GL_REC；接下来调用剩余空间填充算法 LoadLeftSpace($input$)（详见算法 2）完成装箱过程，优层集合 $BEST\_GL$ 就是最终的装箱结果. 本文总共包含 7 个算法（其中算法 3 包含两个部分，分别以算法 3(a)和算法 3(b)表示），算法之间的调用关系如图 4 所示，对于图中任意两个算法 m 和 n，如果算法 n 的框被包含在算法 m 的框中，则表示算法 m 直接调用算法 n.

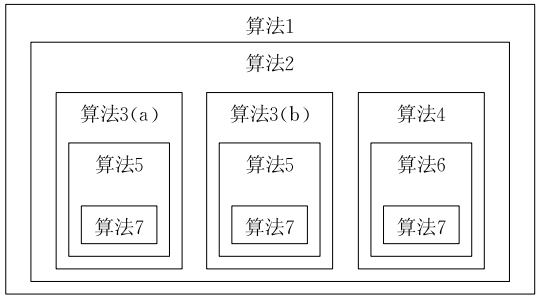

图 4　算法调用关系

算法 1.　启发式正交二叉树搜索算法.

HeuristicOrthogonalBinaryTreeSearch(C, B, fr_gate)

生成空的优层集合 GL

定义全局变量 $BEST\_GL := GL$

定义 $x\_left := l_C$

定义 $y\_left:=w_C$

生成空的优条背包记录集合 GS_REC

生成空的优层背包记录集合 GL_REC

$\text{LoadLeftSpace}\begin{pmatrix}GL,B,x\_left,y\_left,fr\_gate,\\GS\_REC,GL\_REC,1\end{pmatrix}$

$\text{LoadLeftSpace}\begin{pmatrix}GL,B,x\_left,y\_left,fr\_gate,\\GS\_REC,GL\_REC,2\end{pmatrix}$

返回 $BEST\_GL$

剩余空间填充算法的流程如算法 2 所示，该算法为递归算法. ① 判断箱子是否装完或者容器是否已经容不下新的箱子，如果是，计算结束，如果否转到步②；② 调用优条集合生成算法 Make_GS1($input$)(见算法 3(a))和 Make_GS2($input$)(见算法 3(b))将所有箱子组合成优条集合 X_GS；③ 如果 X_GS 为空集则计算结束，否则转到④；④ 调用优层生成算法 MakeGoodLayer($input$)(详见算法 4)生成 x 方向最优层 x_gl；⑤ 生成空优层集合 X_GL，将 GL 中所有优层和 x_gl 拷贝给 X_GL；⑥ 判断 X_GL 中所有箱子体积和是否大于 $BEST\_GL$ 中所有箱子体积和，如果是，则将 X_GL 值赋给 $BEST\_GL$；⑦ 从 X_GS 中去除 x_gl 中包含的所有优条，创建空的箱子集合 X_B，将 X_GS 中包含的所有箱子拷贝至 X_B；⑧ 递归调用算法 2；接下来在 y 方向上重复步②至步⑧，这里不再一一赘述.

算法 2. 剩余空间填充算法.

LoadLeftSpace($GL,B,x\_left,y\_left,fr\_gate,GS\_REC,GL\_REC,n$)

if($B\neq\varnothing$并且存在 $b_i\in B$ 可放入容器 C 剩余空间)

$\text{if}(n=1)\begin{Bmatrix}X\_GS\\Y\_GS\end{Bmatrix}:=\text{Make\_GS1}\begin{pmatrix}B,C,x\_left,y\_left,\\GS\_REC,fr\_gate\end{pmatrix}$

$\text{if}(n=2)\begin{Bmatrix}X\_GS\\Y\_GS\end{Bmatrix}:=Make\_GS2\begin{pmatrix}B,C,x\_left,y\_left,\\GS\_REC,fr\_gate\end{pmatrix}$

if($X_GS\neq\varnothing$并且 $Y\_GS\neq\varnothing$)

//以下为 x 轴方向排列优条

$x\_gl:=MakeGoodLayer(X\_GS,x\_left,GL\_REC)$

创建空集 X_GL，并将 GL 中优层和 x_gl 拷贝给 X_GL

if(X_GL 中箱子体积和大于 $BEST\_GL$ 中箱子体积和)

$BEST\_GL:=X\_GL$ //$BEST\_GL$ 为全局变量

从 X_GS 中去除 x_gl 中包含的所有优条

创建空集 X_B 并将 X_GS 中箱子加入到 X_B 中

LoadLeftSpace($X\_GL,X\_B,x\_left,y\_left-x\_gl.thick,fr\_gate,GS\_REC,GL\_REC,n$)

//以下为 y 轴方向排列优条

$y_gl=$MakeGoodLayer(Y_GS,y_left,GL_REC)

创建空集 Y_GL，并将 GL 中优层和 y_gl 拷贝给 Y_GL

if (Y_GL 中箱子体积和大于 $BEST\_GL$ 中箱子体积和)

$BEST\_GL:=Y\_GL$ //$BEST\_GL$ 为全局变量

从 Y_GS 中去除 y_gl 中包含的所有优条

创建空集 Y_B 并将 Y_GS 中箱子加入到 Y_B 中

LoadLeftSpace ($Y\_GL,Y\_B,x\_left-y\_gl.thick,y\_left,fr\_gate,GS\_REC,GL\_REC,n$)

优条集合 GS 的生成流程分为两种，分别如算法 3(a)和算法 3(b)所示，算法 3(a)和算法 3(b)的区别在于：算法 3(a)利用箱子的三维尺寸中的两个尺寸作为优条的长度和宽度上限，而算法 3(b)只利用箱子的三维尺寸中的一个尺寸同时作为优条的长度和宽度上限，计算结果证明同时调用算法 3(a)和算法 3(b)比只调用其中一个的填充率高. 在算法 3(a)中：① 首先生成空的优条集合 GS；② 然后用所有箱子允许与 xy 面平行的所有面的尺寸轮廓为边界约束生成普通条，普通条生成流程 MakeStrip($input$)如算法 5 所示，一旦找到填充率大于fr_gate 的普通条，就将该普通条作为优条，如果找不到，就选填充率最高的普通条作为优条；③ 并从箱子集合中删除优条中包含的箱子，将优条加入到优条集合；④ 判断箱子集合中是否还有箱子能装入容器剩余空间，如果是，返回到步②执行，如果否，算法结束. 算法 3(b)和算法 3(a)类似，不再赘述.

算法 3(a) 优条集合的生成流程 1.

Generate_GS1($B,C,x\_left,y\_left,GS\_REC,fr\_gate$)

生成空的优条集合 GS

while($B\neq\varnothing$并且存在 $b_i\in B$ 可放入容器 C 剩余空间)

定义临时变量 $best\_gs$ 表示最好的优条

定义临时变量 $gs1,gs2,gs3$ 表示优条

for each box $b\in B$

if(b 允许并能够正放到容器剩余空间)

$gs1:=$MakeStrip($B,h_C,b.l,b.w,GS\_REC,fr\_gate$)

if($fr\_s_{gs1}\geq fr\_gate$)

$best\_gs:=gs1$ 并跳出 for 循环

if(b 允许并能够仰放到容器剩余空间)

$gs2:=$MakeStrip($B,h_C,b.h,b.w,GS\_REC,fr\_gate$)

if($fr\_s_{gs2}\geq fr\_gate$)

$best\_gs:=gs2$ 并跳出 for 循环

if(b 允许并能够侧放到容器剩余空间)

$gs3:=$MakeStrip($B,h_C,b.h,b.l,GS\_REC,fr\_gate$)

if($fr\_s_{gs3}\geq fr\_gate$)

$best\_gs:=gs3$ 并跳出 for 循环

if($gs1,gs2$和$gs3$中最大填充率大于$best_gs$的填充率)

$best\_gs:=$($gs1,gs2$ 和 $gs3$ 中填充率最大者)

从 B 中去除包含在 $best\_gs.Q$ 中的箱子

将 $best\_gs$ 加入到 GS 中

返回 GS

算法 3(b)　优条集合的生成流程 2.

```
Generate_GS2(B,C,x_left,y_left,GS_REC,fr_gate)
  生成空的优条集合 GS
  while(B≠∅并且存在 b_i∈B 可放入容器 C 剩余空间)
    定义临时变量 best_gs 表示最好的优条
    定义临时变量 gs1,gs2,gs3 表示优条
    for each box b∈B
      if(b 允许并能够正放到容器剩余空间空间)
        s1 := max(b.l,b.w)
        gs1 :=MakeStrip(B,h_C,s1,s1,GS_REC,fr_gate)
        if(fr_s_gs1≥fr_gate)
          best_gs :=gs1 并跳出 for 循环
      if(b 允许并能够仰放到容器剩余空间)
        s2 := max(b.w,b.h)
        gs2 := MakeStrip(B,h_C,s2,s2,GS_REC,fr_gate)
        if(fr_s_gs2≥fr_gate)
          best_gs=gs2 并跳出 for 循环
      if(b 允许并能够侧放到容器剩余空间)
        s3 := max(b.l,b.h)
        gs3 := MakeStrip(B,h_C,s3,s3,GS_REC,fr_gate)
        if(fr_s_gs3≥fr_gate)
          best_gs := gs3 并跳出 for 循环
      if(gs1,gs2 和gs3 中最大填充率大于 best_gs 的填充率)
        best_gs := (gs1,gs2 和 gs3 中填充率最大者)
    从 B 中去除包含在 best_gs.Q 中的箱子
    将 best_gs 加入到 GS 中
  返回 GS
```

优层 gl 的生成流程如算法 4 所示. 依次用优条集合 GS 中每一个优条的长度和宽度为即将生成的优层厚度，用 $length$ 为背包容量，以优条集合 GS 为待装入背包的货物，调用算法 MakeLayer($input$)(见算法 6)运算得到一个优层. 比较所有这些优层的层填充率 fr_l，以层填充率 fr_l 最高的优层为运算结果.

算法 4.　优层的生成流程.

```
MakeGoodLayer(GS,length,GL_REC)
  定义临时变量 best_gl 表示最好的优层
  for each gs∈GS
    定义临时变量 gl1,gl2 表示优层
    gl_1 := MakeLayer(GS,length,gs.l_s,GL_REC)
    gl_2 := MakeLayer(GS,length,gs.w_s,GL_REC)
    if(gl_1 或 gl_2 的层填充率大于 best_gl 的层填充率)
      best_gl := gl_1 和 gl_2 中层填充率较大者
  返回 best_gl
```

普通条生成流程如算法 5 所示. 以集合 B 中所有箱子为货物，以容器高度为背包容量，以每个箱子允许在 z 轴方向的最小尺寸为货物重量，以箱子体积为价值，通过动态规划方法解背包问题，得到优条. 在每一次求解背包问题后，都记录本次背包问题的背包容量、优条底面约束尺寸、全部货物列表、选中装入背包的货物列表. 在生成新的优条时，首先判断有无近似记录可供参考，如果有，则直接根据记录生成优条，从而避免重复运算相同背包问题，大大减少装箱计算时间. 查找近似背包问题记录判断流程参见算法 7. 定义优条背包生成记录如下：

$$Knapsack\_Rec = (length, size1, size2, ALL, SELECTED) \quad (12)$$

其中 $length$ 表示背包容量，$size1$ 表示优条第 1 个底面尺寸，$size2$ 表示优条第 2 个底面尺寸，ALL 表示全部货物列表，$SELECTED$ 表示选中装入背包的货物列表.

算法 5.　普通条生成流程.

```
MakeStrip(B,height,max_l,max_w,GS_REC)
  定义 size1 := Max(max_l,max_w)
  定义 size2 := Min(max_l,max_w)
  创建空的箱子列表 B_NEW
  for each box b∈B
    定义 b 对应货物重量 g_wei := 0
    if (b.zh==1 并且 Max(b.l,b.w)≤size1
      并且 Min(b.l,b.w)≤size2)
      g_wei := b.h
    if (b.zw==1 并且 Max(b.l,b.h)≤size1
      并且 Min(b.l,b.h)≤size2
      并且 b.w<g_wei)
      g_wei := b.w
    if (b.zl==1 并且 Max(b.w,b.h)≤size1
      并且 Min(b.w,b.h)≤size2
      并且 b.l<g_wei)
      g_wei := b.l
    if (g_wei>0)//说明箱子 b 可以放进优条中
      将 b 加入到 B_NEW
  定义普通条 strip
  strip :=Exist_Knapsack_Rec(GS_REC,height,max_l,
                             max_w,B_NEW)
  if(strip=∅)
    strip :=(以 B_NEW 为货物列表，以 height 为背包容
          量，以相应 g_wei 为货物重量，以箱子体积为
          货物价值，利用动态规划法求解背包问题)
    Knapsack_Rec := (height,max_l,max_w,B,gs.Q)
                    加入到 GS_REC 中
  返回 strip
```

普通层生成流程如算法 6 所示. 以集合 GS 中所有优条为货物，以 $length$ 为背包容量，以每个优条允许在与优层厚度 $thick$ 垂直方向的最小尺寸为货物重量，以优条中箱子体积和为优条价值，通过动态规划方法解背包问题，得到普通层. 在每一次求解

背包问题后,都记录本次背包问题的背包容量、层厚度、全部货物列表、选中装入背包的货物列表,在进行其他单一层生成时,首先判断有无近似记录可供参考,如果有,则直接根据记录生成优层,从而避免重复运算相同背包问题,大大减少装箱计算时间.查找近似背包记录的判断可以参考算法 7,不再赘述.

算法 6. 单一层生成流程.

```
MakeLayer(GS,length,thick,GL_REC)
  创建空的优条列表 GS_NEW
  for each box gs∈GS
    定义 gs 对应货物重量 g_wei:=0
    定义 size1:=Max(gs.l_s,gs.w_s)
    定义 size2:=Min(gs.l_s,gs.w_s)
    if(size1≤thick)
      g_wei:=size2
    else if(size2≤thick)
      g_wei:=size1
    if(g_wei>0)//说明优条 gs 可以放进优层中
      将 gs 加入到 GS_NEW
  定义普通层 layer
  layer:=Exist_Knapsack_Rec(GL_REC,length,
                            thick,thick,GS)
  if(layer=∅)
    layer:=(以 GS_NEW 为货物列表,以 length 为被
            包容量,以相应 g_wei 为货物重量,以优
            条中箱子体积和为货物价值,利用动态规
            划法求解背包问题)
    Knapsack_Rec:=(length,thick,thickness,GS,gl.T)
    将 Knapsack_Rec 加入到 GL_REC 中
  返回 layer
```

查找近似背包问题记录判断流程如算法 7 所示.轮流访问优条记录集合中每一个优条生成记录,如果记录中优条高度和底面轮廓和给定值相同,并且:①给定箱子集是该记录对应的全部箱子集合的子集;②该记录对应选中的箱子集合是给定箱子集的子集,则认为该记录为近似背包记录,直接返回该记录选中的箱子集即可.

算法 7. 查找近似背包记录流程.

```
Exist_Knapsack_Rec(GS_REC,length,max_l,max_w,
                   ALL)
  for each Knapsack_Rec kr∈GS_REC
    if(kr.length=length,并且max(kr.size1,kr.size2)=
       max(max_l,max_w),并且min(kr.size1,kr.size2)=
       min(max_l,max_w))
      if(ALL⊆kr.ALL 并且 ALL⊇kr.SELECTED)
        返回 kr.SELECTED
  返回∅
```

4 计算实验

本文的二叉搜索树算法(HOBTS)用 C# 语言实现,实验程序运行在 Intel Core2 Q8300@2.50GHz 处理器上,运行环境为 Windows 7 专业版,算法的计算时间主要消耗在形成优条和优层的背包运算上,由于采用了根据近似装箱记录来复制优条,从而大大减少背包运算调用次数,大大减少了装箱计算时间.为了提高装箱率,我们以 0.005 为间距,从 0.9～1 设置 fr_gate 值,每个装箱实例调用算法 1 总共 21 次,从中选择填充率最高的装箱方案作为最终方案.本文的算例数据来源于文献[9],本算法在箱子种类强多样性环境中优势突出,包括 BR1-BR15 一共 1500 个三维装箱实例,他们可以从 OR-Library 网站①下载.这些实例共分为 15 种类型,每类 100 个实例.每类问题中箱子类型数相同,BR1～BR15 中箱子的类型数分别为 3、5、8、10、12、15、20、30、40、50、60、70、80、90、100,多样性由弱到强,能够很好地测试算法在不同多样性装箱问题中的性能.

许多研究者全部或部分测试了文献[9]中的 1500 个实例其中组合启发式算法(CH)[25]、顺序和并行执行的禁忌搜索算法(PTSA)[14]、MFB 算法[23]、随机启发式算法[24]、H_B 算法[32]、启发式算法(SPBBL-CC4)[5]致力于研究弱异构装箱问题,仅测试 BR1～BR7.

H_BR 算法[9]、GA_GB 算法[10]、禁忌搜索算法 TS_BG[11]、贪心随机自适应搜索算法(GRASP)[15]、maximal-space 算法[16]、可变邻域搜索算法(VNS)[17]、混合模拟退火算法(HSA)[26]、整数拆分树搜索算法(CLTRS)[18]、FDA 算法[31]以及最新的多层启发式搜索算法(MLHS)[28]测试了 BR1～BR15 全部实例.A2 算法[27]测试了 BR8～BR15.

4.1 实例计算结果分析

上述比较算法全部满足 C1 约束,部分满足 C2 约束,算法的计算结果直接来自于相应文献.由于本文算法的优势主要体现在箱子种类数较多的情况下,因此本文仅测试了 BR8～BR15.表 2 和表 3 分别列出了算法的 fr_gate 取值详细信息和填充率详细信息.

① OR Library. http://people.brunel.ac.uk/～mastjjb/jeb/orlib/thpackinfo.html

表 2　HOBTS 算法的 *fr_gate* 取值数据

case	*fr_gate*							
	BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15
case1	*1	*0.955	*0.965	*1	*0.94	*0.935	*0.965	*0.935
case2	*0.955	*0.95	*0.965	*0.94	*0.925	*0.925	*0.925	*1
case3	#1	*0.95	*0.92	#0.915	*0.97	*0.945	*0.965	*1
case4	*0.945	*0.915	*0.925	*0.955	*0.945	*0.96	*1	*0.94
case5	*0.96	*0.95	*0.955	#0.97	#0.905	*0.96	*0.965	#0.93
case6	*0.975	*0.98	*0.94	*1	*0.925	*0.98	*0.975	*0.945
case7	#0.925	*0.965	*0.95	*1	*0.95	*0.94	*1	#0.91
case8	*0.955	#0.94	#0.9	*0.935	*0.96	*0.985	#1	*0.935
case9	*0.965	*0.94	*1	*1	*0.945	#0.965	*0.945	*0.945
case10	*0.915	*0.94	*0.925	*0.95	*0.965	*1	*1	*0.965
case11	*0.965	*1	*0.935	*1	*1	*0.945	*1	*0.985
case12	*1	*0.95	*0.955	*1	*0.94	*0.935	*0.94	*0.975
case13	*0.94	*0.945	*0.94	*0.95	*0.965	*1	*0.915	*0.95
case14	*0.935	*0.91	*0.925	#0.945	*0.95	*0.965	*1	*0.925
case15	*0.96	*0.94	*0.95	*0.95	*0.925	*0.95	#0.935	*0.975
case16	*0.9	*1	*1	*0.96	*0.96	*0.96	*0.95	*0.96
case17	#0.925	*0.945	#0.9	*0.93	#0.915	*0.955	*0.955	*0.965
case18	*0.935	*0.92	*0.91	*0.92	#0.945	*0.94	*0.945	*0.93
case19	*0.935	*0.945	*0.935	#1	*0.905	*0.955	*1	*1
case20	*0.96	*1	*1	*0.915	*0.945	*0.96	*0.94	*0.945
case21	*0.95	*0.96	*0.95	*0.95	#0.92	*0.96	*0.935	*0.93
case22	#0.97	*0.97	#0.92	*0.95	*0.96	*0.965	*0.925	*0.965
case23	*0.975	*0.935	*1	*1	*0.96	*1	*0.955	*0.93
case24	#0.945	*0.935	*0.96	*0.93	*0.945	*0.975	*0.95	#0.93
case25	*1	*1	*1	#0.945	*0.955	*1	*0.94	*1
case26	*0.9	*0.95	*0.94	*0.93	#1	*0.96	#0.905	#0.9
case27	*0.94	*0.975	*1	*0.935	*0.95	*0.94	*0.92	*1
case28	#0.93	*0.94	*0.935	*0.96	*1	*0.95	#0.915	*1
case29	*1	*0.95	*0.985	*1	#0.95	*0.935	#0.96	*0.95
case30	*0.965	*0.935	*0.955	*0.98	*1	*0.975	*0.955	*1
case31	*0.955	*0.955	*0.925	*0.96	*0.96	*0.97	*1	*0.92
case32	*0.96	*0.94	#1	*0.945	*0.97	*0.95	*0.93	*0.965
case33	*0.96	*0.96	*0.95	*1	*0.965	*0.925	*0.905	*0.94
case34	*0.98	*0.96	#0.925	*0.945	*0.97	#0.955	*0.96	#0.93
case35	*0.955	*0.91	*0.975	*0.945	#1	*0.94	*0.92	#0.935
case36	*0.97	*0.945	*0.92	#0.91	*0.95	*1	*0.965	*1
case37	*0.96	*0.93	*1	*0.95	*0.96	*1	*0.905	#0.94
case38	*0.93	#0.935	*0.935	*1	*0.955	*0.93	*0.965	#0.955
case39	*0.96	*0.94	*1	*0.955	*0.95	*1	#1	*0.955
case40	*1	*0.975	*0.965	#1	*0.975	*0.935	#0.955	*0.965
case41	#1	*0.95	*1	*0.95	#0.95	#1	*0.945	*0.955
case42	*1	*0.95	*0.935	#1	*0.94	*0.965	*0.965	*0.955
case43	*1	*1	*0.96	*0.935	*0.955	*0.955	*0.96	*0.945
case44	*0.965	*1	*0.96	*0.96	*1	*0.97	*0.935	#0.91
case45	*0.955	*0.935	*0.96	*1	*0.945	*0.945	#0.95	*0.98
case46	#1	*0.9	*0.93	*0.97	*0.94	*1	*0.93	*0.96
case47	*0.95	*0.975	*1	*0.95	*0.97	#0.97	*0.965	*0.92
case48	*0.975	*0.96	*1	#1	*0.955	*0.93	*0.935	*0.95
case49	*1	*1	*0.97	*0.955	*0.945	*0.955	#0.97	*0.965
case50	*0.96	*0.94	#0.905	*1	*0.97	*0.97	*1	*0.96
case51	*1	*0.945	*0.96	*0.955	*1	#1	*0.95	*0.975
case52	*0.935	*0.94	*0.95	*0.965	*1	#1	*0.925	*0.96
case53	*1	*0.935	*0.955	*0.965	*0.95	*0.96	*0.915	*0.98
case54	*0.905	*0.965	*0.92	*0.975	*1	*0.96	#1	*0.97
case55	*1	*0.975	*1	*0.965	*0.945	*0.93	*1	*1
case56	*0.935	*1	*0.965	*1	*0.94	*0.965	*0.93	*0.975
case57	*1	*0.97	#0.925	#0.945	#0.955	#0.915	*0.915	#0.925
case58	*0.935	*0.91	*0.94	*0.965	*0.955	*0.95	*0.955	#0.9
case59	*1	*0.935	*1	#0.96	*0.975	#0.945	*0.93	*0.93
case60	*0.96	*1	*1	*1	*1	#0.91	#0.91	*0.905
case61	*0.935	#0.935	*0.945	*0.95	*1	#1	*0.97	*0.97

（续 表）

case	fr_gate							
	BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15
case62	*0.955	*0.95	*0.95	*1	*1	*0.925	*0.965	*1
case63	*1	*0.955	*0.945	*1	*0.95	*0.965	*1	*0.94
case64	*1	*0.935	*0.965	*0.95	#0.96	*1	*0.935	*0.935
case65	*0.95	*1	*0.94	*0.965	*0.96	*0.955	*0.975	*0.955
case66	*0.97	*0.94	*0.945	*0.965	*0.95	#0.94	*0.97	#0.94
case67	*0.93	*0.94	*1	*0.93	*0.98	*0.965	*0.945	*0.97
case68	*0.935	*0.955	*1	*0.975	*1	*1	*1	#0.93
case69	*0.92	*0.915	*1	#0.975	*1	*1	*1	#0.94
case70	*1	*0.92	#0.95	*0.955	*0.96	*0.935	*0.915	*0.95
case71	*1	*0.94	*0.975	#0.9	*1	#0.945	*0.96	*0.96
case72	*1	*1	*0.98	*0.965	*0.965	*1	*1	*1
case73	*1	#1	*0.955	*0.93	*0.97	#1	*0.97	*0.95
case74	*0.91	#0.945	*0.965	#0.965	*0.93	*0.955	#0.925	#0.97
case75	*0.93	*0.96	*0.94	*0.94	#0.955	*0.97	*1	#0.915
case76	*0.94	*1	*0.96	*0.945	*0.965	*0.945	*1	#0.9
case77	*0.96	*0.95	#0.9	*0.95	*0.93	*0.95	*0.95	*0.93
case78	*0.965	*0.945	*0.945	*1	*0.95	*1	#0.95	#0.9
case79	*0.935	*0.95	*1	*0.97	#0.93	*0.945	*0.96	*0.935
case80	*0.945	*0.9	#0.95	*0.965	*0.96	#1	*0.945	*0.965
case81	*0.93	*0.92	#0.93	*0.915	*1	*0.94	*0.915	*0.945
case82	*0.975	*0.93	*1	*0.935	*0.945	#1	*0.96	*0.94
case83	*0.95	*0.95	*1	*1	*0.93	*0.95	*0.96	*0.94
case84	*0.925	*0.97	*0.945	*0.945	#0.925	*1	*0.96	*0.945
case85	*0.94	#0.915	*0.99	*0.955	*1	*0.955	*0.965	*0.97
case86	*0.945	*0.955	*0.97	*0.965	*0.97	*0.945	*0.96	*1
case87	*0.975	*0.98	*0.98	#0.94	*1	*0.96	*1	*1
case88	*0.95	*0.935	*0.95	*0.95	*1	*0.945	*0.94	*0.95
case89	#0.935	*1	*0.95	*0.98	*1	*1	*1	*0.975
case90	*0.95	*0.96	*1	*1	*1	*1	*0.95	*0.945
case91	*1	*0.925	*1	*0.94	*0.955	#0.905	*0.945	*0.945
case92	#0.94	#0.93	*0.95	*0.96	*0.965	*0.955	*1	*0.955
case93	*1	*0.96	*0.985	*0.97	*0.97	*0.965	*0.95	*0.94
case94	#0.9	*0.945	*1	*0.955	*0.96	*1	*0.98	*0.96
case95	*0.95	*0.97	*0.92	*0.96	*0.955	*0.965	*0.96	*0.945
case96	*0.92	*1	*0.975	*0.945	*0.96	*0.945	*0.96	*1
case97	*1	*1	*0.915	*0.94	*0.965	*0.94	#1	*0.93
case98	*1	*1	*0.94	*0.97	*0.97	*0.975	*1	*0.95
case99	*0.965	*0.945	*0.925	*1	#0.96	*0.945	*0.95	#0.945
case100	*0.96	#0.965	*1	*1	*1	*0.935	*1	#0.9

注：* 指通过调用算法 3(a)得到最优值；# 指通过调用算法 3(b)得到最优解.

表 3 HOBTS 算法的填充率数据

case	填充率							
	BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15
case1	0.90770	0.89982	0.90167	0.89797	0.90258	0.89360	0.90087	0.89307
case2	0.92014	0.91775	0.91323	0.91107	0.90090	0.89556	0.89812	0.89895
case3	0.91217	0.90504	0.90864	0.89615	0.89290	0.90126	0.89008	0.89937
case4	0.88816	0.91346	0.90444	0.90533	0.88879	0.90384	0.90223	0.89597
case5	0.91183	0.91502	0.90883	0.89898	0.89594	0.89527	0.89969	0.89156
case6	0.90661	0.90806	0.89328	0.90282	0.90234	0.89777	0.90404	0.90263
case7	0.90802	0.91340	0.90420	0.89743	0.89290	0.89410	0.89312	0.89097
case8	0.91263	0.89434	0.90085	0.89985	0.89802	0.89907	0.89958	0.89251
case9	0.90662	0.90547	0.89940	0.90570	0.90863	0.90130	0.89385	0.89561
case10	0.89998	0.89705	0.91195	0.90140	0.90265	0.90193	0.89895	0.89438
case11	0.91431	0.90636	0.90511	0.90618	0.91571	0.90283	0.90707	0.90332
case12	0.90958	0.90010	0.89958	0.89808	0.89339	0.90206	0.89752	0.89659
case13	0.91936	0.91310	0.91154	0.90236	0.89594	0.89495	0.89537	0.88397
case14	0.92759	0.90572	0.90999	0.89894	0.89652	0.89396	0.89635	0.89713
case15	0.91454	0.90734	0.91163	0.90778	0.91086	0.90175	0.89618	0.89788
case16	0.91608	0.91793	0.91070	0.90103	0.89784	0.89827	0.89151	0.90004
case17	0.90342	0.90217	0.90042	0.90790	0.90157	0.90257	0.89997	0.89526

（续 表）

case	填充率							
	BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15
case18	0.90595	0.91068	0.89770	0.89768	0.89584	0.89255	0.89052	0.88843
case19	0.90778	0.90540	0.91364	0.90159	0.90700	0.90599	0.89023	0.89687
case20	0.91886	0.91769	0.91389	0.90792	0.90602	0.89965	0.89896	0.89637
case21	0.90497	0.89765	0.89974	0.89170	0.89440	0.89567	0.89494	0.88871
case22	0.92073	0.91164	0.91157	0.90767	0.89934	0.89390	0.89121	0.89035
case23	0.91044	0.90876	0.90154	0.89391	0.90093	0.90125	0.89267	0.89252
case24	0.90921	0.90727	0.90766	0.90367	0.89880	0.90092	0.89726	0.89330
case25	0.92362	0.90578	0.90503	0.90535	0.90448	0.89771	0.88962	0.90157
case26	0.89632	0.90858	0.89898	0.88820	0.88825	0.88995	0.87980	0.88533
case27	0.91687	0.90514	0.90367	0.90726	0.90332	0.89254	0.89415	0.88973
case28	0.91171	0.91289	0.89801	0.90546	0.89177	0.89057	0.89178	0.89422
case29	0.90845	0.90423	0.89981	0.89831	0.89417	0.90826	0.90062	0.89547
case30	0.91121	0.91176	0.90876	0.90804	0.91777	0.90554	0.89452	0.90019
case31	0.91369	0.90751	0.89698	0.90180	0.89162	0.90181	0.89502	0.89303
case32	0.92269	0.90786	0.91129	0.90512	0.90884	0.90122	0.89106	0.88977
case33	0.92246	0.91239	0.91227	0.90377	0.90440	0.89784	0.89078	0.90375
case34	0.91677	0.91237	0.90159	0.90239	0.89999	0.88911	0.89440	0.89042
case35	0.92450	0.92124	0.91881	0.90345	0.90109	0.90628	0.90427	0.90843
case36	0.91790	0.90566	0.90842	0.90477	0.90332	0.90361	0.90132	0.89417
case37	0.91375	0.91363	0.91069	0.90764	0.91141	0.90166	0.89762	0.89343
case38	0.90215	0.90151	0.91094	0.89517	0.90376	0.89619	0.90075	0.89578
case39	0.91590	0.91043	0.90747	0.90761	0.89736	0.89297	0.89610	0.89472
case40	0.90973	0.90132	0.90317	0.90366	0.90057	0.89338	0.88523	0.88772
case41	0.91600	0.91946	0.91403	0.90666	0.90376	0.89453	0.89171	0.88694
case42	0.90305	0.90716	0.91595	0.91176	0.90853	0.90581	0.89914	0.89519
case43	0.91039	0.91369	0.90374	0.91366	0.90383	0.89296	0.89443	0.88214
case44	0.91825	0.91096	0.90749	0.90847	0.90485	0.91134	0.90077	0.89494
case45	0.90842	0.90414	0.89709	0.89433	0.89798	0.87981	0.89315	0.88599
case46	0.91159	0.90203	0.89483	0.89705	0.90405	0.89433	0.89590	0.89381
case47	0.90600	0.89462	0.90103	0.88284	0.89651	0.89862	0.89743	0.88575
case48	0.91649	0.91364	0.91801	0.91095	0.91072	0.90439	0.90145	0.89678
case49	0.91397	0.90566	0.91034	0.89297	0.90514	0.89949	0.89287	0.89014
case50	0.91000	0.91832	0.89750	0.89802	0.89840	0.88968	0.89859	0.89089
case51	0.91986	0.92402	0.91690	0.90557	0.90930	0.90258	0.90063	0.88785
case52	0.91688	0.91301	0.90282	0.90468	0.89899	0.89723	0.89580	0.89643
case53	0.92337	0.91166	0.90521	0.89502	0.90343	0.90033	0.89960	0.89009
case54	0.89335	0.90457	0.90254	0.90257	0.89705	0.90270	0.89200	0.89517
case55	0.91449	0.90386	0.90815	0.90534	0.90909	0.90450	0.89779	0.89523
case56	0.90122	0.91793	0.91147	0.91137	0.89774	0.90882	0.89962	0.90009
case57	0.91399	0.91630	0.90075	0.89580	0.90256	0.89017	0.90023	0.89076
case58	0.91339	0.91077	0.91701	0.90710	0.90232	0.90683	0.90170	0.90090
case59	0.90034	0.90761	0.90465	0.89727	0.90477	0.89724	0.90175	0.89566
case60	0.91190	0.90940	0.91182	0.90953	0.91201	0.89753	0.89307	0.89395
case61	0.91893	0.90755	0.91611	0.90428	0.91387	0.90575	0.90597	0.89392
case62	0.89910	0.91517	0.90641	0.90153	0.89956	0.89092	0.89732	0.89359
case63	0.90567	0.90948	0.89652	0.89430	0.90061	0.90261	0.89955	0.89353
case64	0.91332	0.90781	0.90110	0.90091	0.88563	0.89453	0.89109	0.89127
case65	0.92422	0.91656	0.90493	0.91606	0.90667	0.90132	0.90452	0.91461
case66	0.90972	0.89964	0.89734	0.89690	0.90731	0.90127	0.89709	0.89797
case67	0.91838	0.90249	0.90137	0.89905	0.89860	0.89424	0.88649	0.88453
case68	0.92846	0.90592	0.91243	0.90511	0.90911	0.90359	0.90216	0.90132
case69	0.90357	0.89773	0.89907	0.89424	0.90339	0.89012	0.89158	0.88314
case70	0.90982	0.90382	0.89325	0.89821	0.90816	0.90289	0.89397	0.89253
case71	0.90838	0.92125	0.92109	0.91466	0.91133	0.90618	0.90509	0.89587
case72	0.90858	0.90409	0.90305	0.90594	0.90360	0.90583	0.89263	0.89041
case73	0.90799	0.91213	0.90416	0.90097	0.90924	0.89868	0.89925	0.88871
case74	0.91332	0.89957	0.89867	0.90027	0.90356	0.89677	0.88966	0.89284
case75	0.89662	0.90410	0.90500	0.91715	0.90020	0.89831	0.89463	0.88965
case76	0.91023	0.90719	0.90353	0.90071	0.91155	0.89996	0.89060	0.88511
case77	0.91539	0.90831	0.91570	0.89765	0.89794	0.90184	0.88961	0.89421
case78	0.91588	0.91122	0.91319	0.90625	0.89399	0.89509	0.89074	0.88542

（续 表）

case	填充率							
	BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15
case79	0.89521	0.89485	0.89161	0.89797	0.89902	0.89185	0.89636	0.89523
case80	0.91225	0.90608	0.91030	0.90627	0.90380	0.89835	0.90031	0.89759
case81	0.89287	0.88967	0.89860	0.88758	0.88509	0.88501	0.88400	0.87546
case82	0.90024	0.89656	0.90293	0.89952	0.88423	0.88429	0.88738	0.89374
case83	0.92198	0.91749	0.90937	0.90997	0.90016	0.89740	0.89824	0.89908
case84	0.91051	0.90928	0.90696	0.90613	0.89251	0.89564	0.88958	0.88846
case85	0.91403	0.89897	0.90853	0.91428	0.90618	0.90160	0.90053	0.89408
case86	0.92018	0.91650	0.90640	0.90037	0.90079	0.89878	0.90753	0.89135
case87	0.92955	0.91707	0.91732	0.90984	0.91596	0.91213	0.90285	0.90305
case88	0.90956	0.91096	0.90151	0.90374	0.89074	0.89781	0.88906	0.88886
case89	0.90966	0.91028	0.90471	0.91276	0.91012	0.90291	0.89932	0.89711
case90	0.90352	0.91025	0.89934	0.90029	0.89875	0.90043	0.89532	0.89532
case91	0.90970	0.90973	0.90049	0.89396	0.89771	0.89670	0.89159	0.89278
case92	0.90790	0.90905	0.90799	0.90843	0.91075	0.89999	0.89820	0.90109
case93	0.91020	0.91856	0.91948	0.91193	0.90740	0.89459	0.89559	0.89690
case94	0.92893	0.91951	0.91554	0.91546	0.91118	0.90598	0.90369	0.90240
case95	0.91511	0.90652	0.89799	0.89662	0.89047	0.89605	0.89590	0.89783
case96	0.91234	0.90918	0.89944	0.89412	0.89253	0.89477	0.89630	0.88416
case97	0.91035	0.89733	0.90087	0.90098	0.89412	0.89774	0.89845	0.89173
case98	0.92038	0.91389	0.90970	0.90729	0.90339	0.89671	0.89238	0.89293
case99	0.91410	0.89725	0.89532	0.88739	0.89157	0.88956	0.87895	0.89179
case100	0.91506	0.91171	0.91130	0.90779	0.90111	0.89976	0.90534	0.89223

表 4 列出了前述算法与本文算法 HOBTS 的计算结果，表中所有的数据表示每个算法针对一类实例所得到的平均填充率(%). HOBTS 算法的计算结果是在同时满足 C1、C2 和 C3 约束的条件下得出的.

从表 4 可以看出，在同时满足 C1、C2 和 C3 约束的前提下，在箱子种类数在 90 种以上时，填充率超过了目前已知的所有算法. 其中箱子种类数为 90 时，超过目前最好的 MLHS 算法 0.27%. 当箱子种类数为 100 时，比目前最好的 MLHS 算法提高了 0.84%. 当箱子种类数为 60 时，超过 CLTRS 算法 0.17%；当箱子种类数为 70 时，超过 CLTRS 算法 0.63%；当箱子种类数为 80 时，超过 CLTRS 算法 0.87%；当箱子种类数为 90 时，超过 CLTRS 算法 1.35%；当箱子种类数为 100 时，超过 CLTRS 算法 1.81%. 从 HOBTS 算法对于 BR8～BR15 的装箱率趋势来看，箱子种类数越多，越能体现 HOBTS 算法的优势.

表 4 各种算法对 BR8～BR15 的填充率比较

算法	约束	填充率(%)							
		BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15
H_BR[9]	C1&C2	80.10	78.03	76.53	75.08	74.37	73.56	73.37	73.38
GA_GB[10]	C1&C2	87.52	86.46	85.53	84.82	84.25	83.67	82.99	82.47
TS_BG[11]	C1&C2	87.11	85.76	84.73	83.55	82.79	82.29	81.33	80.85
GRASP[15]	C1	90.26	89.50	88.73	87.87	87.18	86.70	85.81	85.48
maximal-space[16]	C1	91.02	90.46	89.87	89.36	89.03	88.56	88.46	88.36
HSA[26]	C1&C2	90.56	89.70	89.06	88.18	87.73	86.97	86.16	85.44
A2[27]	C1	88.41	88.14	87.90	87.88	87.92	87.92	87.82	87.73
VNS[17]	C1	92.78	92.19	91.92	91.46	91.20	91.11	90.64	90.38
CLTRS[18]	C1	93.70	93.44	93.09	92.81	92.73	92.46	92.40	92.40
	C1&C2	92.26	91.48	90.86	**90.11**	**89.51**	**88.98**	**88.26**	**87.57**
FDA[31]	C1	92.92	92.49	92.24	91.91	91.83	91.56	91.30	91.02
MLHS[28]	C1	94.54	94.14	93.95	93.61	93.38	93.14	93.06	92.90
	C1&C2	93.12	92.48	91.83	91.23	90.59	89.99	**89.34**	**88.54**
HOBTS	C1&C2&C3	91.18	90.84	90.59	**90.28**	**90.14**	**89.85**	**89.61**	**89.38**

表 5 给出了本文算法对 BR8～BR15 的计算时间，对 BR8～BR15，HOBTS 算法的平均运行时间是 175.49s，而 CLTRS 和 MLHS 算法在考虑 C1 和 C2 约束(比本文算法少了 C3 约束)，BR1～BR15 的平均计算时间分别为 197.33s 和 320s，由于 BR1～BR7 箱子种类数少，计算时间比 BR8～BR15 平均要少，所以 CLTRS 和 MLHS 算法对于 BR8～BR15 的平均运算时间比 BR1～BR15 的还要多，因此本文算法在计算时间上有优势的.

表 5　本文算法对 BR8～BR15 的计算时间

算例	平均计算时间/s	算例	平均计算时间/s
BR8	83.62	BR12	192.81
BR9	104.78	BR13	224.88
BR10	134.16	BR14	237.37
BR11	157.55	BR15	268.72
平均值	175.49		

4.2　算法的复杂性分析

以最极端情况下的装箱实例为例说明算法的复杂性. 在最极端情况下，任意两个箱子的尺寸都不相同，每个箱子的长宽高都不相同，且可以正放，侧放和仰放. 在分析之前先作如表 6 定义(之前的符号定义在接下来的部分中被自动忽略).

表 6　复杂性分析变量定义

变量名称	变量含义
b_n	箱子数量
$\overline{b\_lwh}$	所有箱子尺寸的平均值
$\overline{c\_lw}$	容器长和宽的平均值
c_h	容器高度
$tree\_layer\_n$	HOBTS 算法的二叉树的层数
$tree\_node\_n$	HOBTS 算法二叉树的节点数
$\overline{gl\_n}$	所有装箱方案中优层的平均数目
gl_com_n	HOBTS 求解一个问题时总共求解优层数
$HOBTS\_T$	HOBTS 求解一个问题的总计算时间
fr_gate_n	使用的 fr_gate 值的数量
$\overline{gl\_t}$	生成一个优层的平均计算时间
$\overline{dp\_gl\_t}$	用动态规划生成一个优层的平均计算时间
$\overline{dp\_gs\_t}$	用动态规划生成一个优条的平均计算时间
$\overline{dp\_u}$	动态规划每迭代一步的平均计算时间

则有 $\overline{gl\_n}\approx\overline{c\_lw}/\overline{b\_lwh}$ 和 $tree\_layer\_n=\overline{gl\_n}+1$. 所以 $tree\_layer\_n\approx\overline{c\_lw}/\overline{b\_lwh}+1$. 又对于二叉树，有 $tree\_node\_n\approx 2^{tree\_layer\_n}-1$，所以 $tree\_node\_n\approx 2^{\overline{c\_lw}/\overline{b\_lwh}+1}-1$，由于二叉树上除了根节点外，每一个节点生成一个新优层，所以 $gl\_com\_n=tree\_node\_n-1$，所以

$$gl\_com\_n\approx 2^{\overline{c\_lw}/\overline{b\_lwh}+1}-2 \tag{13}$$

而算法的时间主要花费在生成优层上. 易得每生成一个优层前，平均剩余箱子数为 $b\_n/2$，

又每生成一个优层，需要生成一组优条，又每个优条平均包含 $c\_h/\overline{b\_lwh}$ 个箱子，所以这组优条平均包含 $(b\_n\times\overline{b\_lwh})/(2\times c\_h)$ 个优条. 为了从优条生成优层，平均需要求解背包运算次数即为优条数目的两倍(以每个优条的长和宽为厚度生成优层)：

$$(b\_n\times\overline{b\_lwh})/c\_h \tag{14}$$

每个动态规划方法求解优层背包问题的计算时间：

$$\overline{dp\_gl\_t}=\frac{\overline{c\_lw}\times(b\_n\times\overline{b\_lwh})}{2\times c\_h}\times\overline{dp\_u} \tag{15}$$

对于特定优层，每生成一个优条前，平均剩余箱子数为优层平均剩余箱子数的一半，即 $b\_n/4$，则可以得到为了从箱子生成优条而调用的背包运算次数为

$$b\_n\times 5/4 \tag{16}$$

其中算法 3(a)调用 $b\_n\times 3/4$ 次，算法 3(b)调用 $b\_n\times 2/4$ 次.

每个动态规划方法求解优条背包问题的计算时间：

$$\overline{dp\_gs\_t}=c\_h\times\frac{b\_n}{4}\times\overline{dp\_u} \tag{17}$$

依据式(14)～(17)可得

$$\overline{gl\_t}=\left(\frac{b\_n^2\times\overline{b\_lwh}^2\times\overline{c\_lw}}{2\times c\_h^2}+\frac{5\times c\_h\times b\_n^2}{16}\right)\times\overline{dp\_u} \tag{18}$$

依据式(13)和(18)，HOBTS 总共求解一维背包问题耗费时间：

$$\begin{aligned}HOBTS\_T&\approx fr\_gate\_n\times gl\_com\_n\times\overline{gl\_t}\\&\approx fr\_gate\_n\times b\_n^2\times(2^{\overline{c\_lw}/\overline{b\_lwh}}-1)\times\left(\frac{\overline{b\_lwh}^2\times\overline{c\_lw}}{c\_h^2}+\frac{5\times c\_h}{8}\right)\times\overline{dp\_u}\end{aligned} \tag{19}$$

所以本文算法的时间复杂度表示为

$$T(b\_n)=O(b\_n^2) \tag{20}$$

我们只在张德富等人[30]的文献中找到对算法的时间复杂度的分析，该文算法的时间复杂度为 $O(b\_n^3)$，说明了本文算法的时间复杂度是有竞争力的.

算法运行过程中，始终只保留一个最优装箱方案，算法 2 是递归算法，其嵌套深度约等于 $\overline{c\_lw}/\overline{b\_lwh}$，该值通常为个位数，算法运行过程中占用的存储空间不大，现今的普通计算机内存完全可以胜任.

以上分析没有考虑算法 7 对整个算法的加速作用，实际计算过程中，调用算法 7 比不调用算法 7 快数倍以上.

4.3　算法的收敛性分析

HOBTS 算法的流程主要表现为算法 2 的递归调用，每一次递归调用之后，容器中增加一个优层，容器中箱子的体积和增加，容器填充率向填充上限(100%)靠近，递归调用次数严格小于容器长宽的较大值与所有箱子尺寸最小值的比值. 在算法的实现函数中，优层数量作为自变量是离散的并且有限的，填充率作为函数值是递增的并且有上限，所以本文

算法是收敛的.

4.4 算法的可行性和稳定性

BR8～BR15 的运行结果说明了本文算法是可行的,式(19)可以看出本文算法的计算时间与箱子数目的平方成线性关系,与 *fr_gate_n* 成线性关系,与容器长宽尺寸对箱子平均尺寸的比值成指数关系.当箱子平均尺寸变化不大时,算法是比较稳定的.当箱子平均尺寸剧烈变小时,计算时间剧烈增加,因此还需要进一步提高算法的稳定性来应对箱子尺寸变小的情况.

5 结束语

三维装箱问题是经典且实用的问题,在众多学者的耕耘下,填充率不断接近极限最优,要提高越来越困难.本文提出了三维装箱问题的启发式二叉树搜索算法,首先将所有箱子组合成多个优条,通过沿容器长度边和宽度边增加一层优条来得到每个节点的左右子节点.二叉树一直扩展,直至所有叶子节点都摆不下任意剩余的箱子为止.最后从所有叶子节点中选择填充率最高的装填方案作为最终的计算结果.HOBTS 算法首次可以同时满足 C1、C2 和 C3 三大约束.在箱子种类数小于 90 时,平均填充率略低于 MLHS 算法,在箱子种类数小于 60 时,平均填充率略低于 CLTRS 算法,但是在箱子种类数超过 90 时,HOBTS 算法最优,通过增加 *fr_gate* 值的密集度,填充率还可以提升.另外 HOBTS 算法也存在需要改进的地方,当箱子种类数较少时,HOBTS 算法的装箱率相对于现有算法不具有优势;另外二叉树搜索深度取决于摆放的优层数,当箱子体积相对于容器容积越小时,优层数就越多,二叉树搜索深度就越深,计算时间就越长.

参 考 文 献

[1] Dyckhoff H, Finke U. Cutting and Packing in Production and Distribution. Heidelberg: Physica-Verlag, 1992

[2] Wäscher G, Haußner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research, 2007, 183(3): 1109-1130

[3] Bischoff E E, Marriott M D. A comparative evaluation of heuristics for container loading. European Journal of Operational Research, 1990, 44(2): 267-276

[4] Lu Yi-Ping, Zha Jian-Zhong. Sequence triplet method for 3D rectangle packing problem. Journal of Software, 2002, 13(11): 2183-2187(in Chinese)
(陆一平, 查建中. 三维矩形块布局的序列三元组编码方法. 软件学报, 2002, 13(11): 2183-2187)

[5] Bortfeldt A, Mack D. A heuristic for the three dimensional strip packing problem. European Journal of Operational Research, 2007, 183(3): 1267-1279

[6] Faroe O, Pisinger D, Zachariasen M. Guided local search for three-dimensional bin-packing problem. INFORMS Journal on Computing, 2003, 15(3): 267-283

[7] Pisinger D. Heuristics for the container loading problem. European Journal of Operational Research, 2002, 141(2): 143-153

[8] George J A, Robinson D F. A heuristic for packing boxes into a container. Computers and Operations Research, 1980, 7(3): 147-156

[9] Bischoff E E, Ratcliff B S W. Issues in the development of approaches to container loading. Omega, 1995, 23(4): 377-390

[10] Gehring H, Bortfeldt A. A genetic algorithm for solving the container loading problem. International Transactions in Operational Research, 1997, 4(5-6): 401-418

[11] Bortfeldt A, Gehring H. A tabu search algorithm for weakly heterogeneous container loading problems. OR Spectrum, 1998, 20(4): 237-250

[12] Bortfeldt A, Gehring H. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research, 2001, 131(1): 143-161

[13] Eley M. Solving container loading problems by block arrangement. European Journal of Operational Research, 2002, 141(2): 393-409

[14] Bortfeldt A, Gehring H, Mack D. A parallel tabu search algorithm for solving the container loading problem. Parallel Computing, 2003, 29(5): 641-662

[15] Moura A, Oliveira J F. A GRASP approach to the container-loading problem. IEEE Intelligent Systems, 2005, 20(4): 50-57

[16] Parreno F, Alvarez-Valdes R, Oliveira J F, Tamarit J M. A maximal space algorithm for the container loading problem. INFORMS Journal on Computing, 2007, 20(3): 412-422

[17] Parreno F, Alvarez-Valdes R, Oliveira J F, Tamarit J M. Neighborhood structures for the container loading problem: AVNS implementation. Journal of Heuristics, 2010, 16(1): 1-22

[18] Fanslau T, Bortfeldt A. A tree search algorithm for solving the container loading problem. INFORMS Journal on Computing, 2010, 22(2): 222-235

[19] Ngoi B K A, Tay M L, Chua E S. Applying spatial representation techniques to the container packing problem. International Journal of Production Research, 1994, 32(1): 111-123

[20] Morabito R, Arenales M. An AND/OR-graph approach to the container loading problem. International Transactions in Operational Research, 1994, 1(1): 59-73

[21] Sixt M. Dreidimensionale Packprobleme. Losungsverfahren

basierend auf den Meta-Heuristiken Simulated Annealing und Tabu-Suche. Frankfurt am Main: Europaischer Verlag der Wissenschaften, 1996

[22] Gehring H, Bortfeldt A. A parallel genetic algorithm for solving the container loading problem. International Transactions in Operational Research, 2002, 9(4): 497-511

[23] Lim A, Rodrigues B, Yang Y. 3-D container packing heuristics. Applied Intelligence, 2005, 22(2): 125-134

[24] Juraitis M, Stonys T, Starinskas A, et al. A randomized heuristic for the container loading problem: Further investigations. Information Technology and Control, 2006, 35(1): 7-12

[25] He Da-Yong, Zha Jian-Zhong, Jiang Yi-Dong. Research on solution to complex container loading problem based on genetic algorithm. Journal of Software, 2001, 12(9): 1380-1385(in Chinese)
(何大勇，查建中，姜义东. 遗传算法求解复杂集装箱装载问题方法研究. 软件学报，2001，12(9)：1380-1385)

[26] Zhang De-Fu, Peng Yu, Zhu Wen-Xing, Chen Huo-Wang. A hybrid simulated annealing algorithm for the three-dimensional packing problem. Chinese Journal of Computers, 2009, 32(11): 2147-2156(in Chinese)
(张德富，彭煜，朱文兴，陈火旺. 求解三维装箱问题的混合模拟退火算法. 计算机学报，2009，32(11)：2147-2156)

[27] Huang Wen-Qi, He Kun. A caving degree approach for the single container loading problem. European Journal of Operational Research, 2009, 196(1): 93-101

[28] Zhang De-Fu, Peng Yu, Zhang Li-Li. A multi-layer heuristic search algorithm three dimensional container loading problem. Chinese Journal of Computers, 2012, 35(12): 2553-2561(in Chinese)
(张德富，彭煜，张丽丽. 求解三维装箱问题的多层启发式搜索算法. 计算机学报，2012，35(12)：2553-2516)

[29] Li Guang-Qiang, Teng Hong-Fei. Isomorphic and non-isomorphic layout patterns of packing problems. Chinese Journal of Computers, 2003, 26(10): 1248-1254(in Chinese)
(李广强，滕弘飞. 装填布局的同构和非同构模式. 计算机学报，2003，26(10)：1248-1254)

[30] Zhang De-Fu, Wei Li-Jun, Chen Qing-Shan, Chen Huo-Wang. A combinational heuristic algorithm for the three dimensional packing problem. Journal of Software, 2007, 18(9): 2083-2089(in Chinese)
(张德富，魏丽军，陈青山，陈火旺. 三维装箱问题的组合启发式算法. 软件学报，2007，18(9)：2083-2089)

[31] He Kun, Huang Wen-Qi. An efficient placement heuristic for three-dimensional rectangular packing. Computers & Operations Research, 2011, 38(1): 227-233

[32] Bischoff E E. Three-dimensional packing of items with limited load bearings trength. European Journal of Operational Research, 2006, 168(3): 952-966

LIU Sheng, born in 1978, Ph. D., associate professor. His research interests include combinatorial optimization, intelligent logistics, intelligent transportation and service computing.

ZHU Feng-Hua, born in 1976, Ph. D., associate professor. His research interests include intelligent transportation, intelligent logistics.

LV Yi-Sheng, born in 1983, Ph. D., assistant researcher. His research interests include intelligent transportation, combinatorial optimization.

LI Yuan-Tao, born in 1979, Ph. D., assistant researcher. His research interests include intelligent transportation, combinatorial optimization.

Background

The work is supported by the National Nature Science Foundation of China (Grant Nos. 61104054, 71232006, and 61233001). The projects aim at designing highly efficient algorithms for NP hard optimization problems, such as the packing problem and cutting problem, which are the most important part of automated packing and cutting software. The objective of this paper was to investigate three dimensional packing problems and to develop high efficient algorithms that could be used in an industrial field. The paper belongs to one of the key parts of the projects.