基于动作空间的三维装箱问题的确定性高效率求解算法

何 琨 黄文奇

(华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 三维装箱问题要求将有限个三维矩形物体尽可能多地装入到一个三维矩形箱子中,使得箱子的填充率即体积利用率最大.在求解三维装箱问题的穴度算法的基础之上,进一步做了以下改进:(1)将当前剩余空间中可能放入的每个体积最大的三维矩形虚拟物体所对应的空间定义为动作空间,在动作空间内放入物体并使穴度的定义体现放入物体与动作空间的吻合程度;(2)在物体放入位置的选择上直接体现"金角银边草肚皮"的思想,每一步只选择最靠近箱子边缘的一个动作空间来装载物体;(3)结合捆绑策略,将形状大小相同的物体捆绑为一个较大的矩形块进行放入,对捆绑块形状大小的选择为在不超出动作空间的前提下尽量用物体填满该空间的两至三个维度.实验结果表明,改进后的穴度算法在付出很少的开销代价的情况下显著地提高了箱子的填充率.

关键词 三维布局;装箱;启发式;动作空间;穴度

中图法分类号 TP301 **DOI 号** 10.3724/SP.J.1016.2014.01786

An Action Space Based Deterministic Efficient Algorithm for Solving the Three-Dimensional Container Loading

HE Kun HUANG Wen-Qi

(*College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract The three-dimensional (3D) container loading problem is a problem of loading a subset of 3D rectangular items into a 3D rectangular container, such that the stowed percentage, i.e. the container's volume utilization, is maximized. Based on the caving degree approach proposed for the 3D container loading problem, we present three improvement strategies in this paper. Firstly, "action space" is defined for an empty space that a maximal dummy 3D rectangular item could be placed. Secondly, an action space nearest to the edge of the container is selected to stow items at each iteration step. In this way the nature of "gold corner, silver side and strawy void" is embodied directly on the candidate packing place. Finally, we bind items in the same size into a larger rectangular block and place the block into a selected action space. The size of the block is determined in such a way that two or three dimensions of the action space are stowed as fully as possible without beyond the boundary of the action space. Then, "caving degree" is defined to reflect to which degree the block dovetail with the space. Experimental results suggest that the improved caving degree algorithm increased the container's volume utilization markedly within a short running time, and it outperforms many excellent algorithms published formally.

Keywords three-dimensional packing; container loading; heuristic; action space; caving degree

收稿日期:2012-03-14;最终修改稿收到日期:2014-04-10.本课题得到国家自然科学基金(61173180)资助.**何 琨**,女,1972 年生,博士,副教授,中国计算机学会(CCF)高级会员,主要研究方向为 NP 难问题的现实求解、数据挖掘.E-mail:brooklet60@gmail.com.**黄文奇**,男,1938 年生,教授,博士生导师,主要研究领域为 NP 难问题的现实求解.

1 引　言

Packing 问题(布局问题)是高复杂度的典型的 NP 难度问题,在现实生活中有着广泛的应用.本文研究三维欧氏空间中一种典型的 Packing 问题——装箱问题(container loading problem),即已知一个形状大小任意给定的长方体形的箱子和有限个形状大小分别任意给定的长方体形的物体,要求确定一个可行的装箱方案,使得箱子的填充率即{(∑放入物体的体积)/箱子的体积×100%}最大.物体的放置要求满足以下约束:放入的物体完全被包含在箱子内,棱平行于箱子的棱,且任意两物体均无重叠.

在实际应用中,特定的装箱问题还需要考虑一些其他的约束,例如方向约束和稳定性约束.本文考虑方向约束,即物体的某些放置方向被禁止使用.在现实生活中,当待放物体内为装满液体的瓶子时会有此类约束.本文不考虑稳定性约束,其原因在于:(1)国内外文献中对稳定性约束的定义并不一致,有的要求每个物体必须得到其他物体或箱子底部的完全支撑,有的要求被支撑的底面积达到一定的比率,有的则仅要求物体的几何重心得到支撑即可;(2)当箱子的填充率很高时,稳定性会成为高密度放置的一种自然的结果;(3)可以在少量的空隙中填入海绵、橡胶或其他填充物以保证整个装载的稳定.

从国内外的相关研究中可看出,启发式方法已成为求解三维装箱这一 NP 难度问题的有效方法.目前国内外研究者多采用构造型启发式算法与邻域搜索算法相结合的方式来求解此问题.其中构造型算法多借助人类在砌砖、砌墙、货物堆放及下围棋等方面的实践经验,求解方法主要包括砌墙法[1-9]、块排列法[10-12]、极大空间法[13-14]、堆构造法[15]、穴度法[16]等.在此基础上,可进一步结合遗传[3-4,15]、退火[5,9,11]、禁忌[5,10,12]、树搜索[6,16]、随机搜索[7-8,13]、变结构邻域搜索[14]等算法以进行邻域搜索.

基于拟人途径,在求解二维矩形 Packing 问题的穴度算法的基础之上[17],我们提出了求解三维装箱问题的穴度算法[16],通过"穴度"来统一评价当前物体与其他已放入物体及箱子的紧密程度,并取得了很好的计算结果.然而,穴度算法的计算时间较长.例如,对于有 100 个形状大小不同的待放物体的装箱实例,其计算时间约为 2 h～3 h.这使得沿原方案进一步改进算法的性能变得比较困难.另外,穴度算法尚未利用形状大小有较多相同的弱异构型算例的特点,因此对弱异构型算例的求解质量不够高.

本文受极大空间法的启发[13-14],将当前剩余空间中每个体积最大的三维矩形空间定义为动作空间,给出了动作空间的明确定义,并改进穴度的定义使之体现放入物体与动作空间的吻合程度.然后,进一步结合捆绑策略以利用弱异构型算例的特点,从而得到了基于动作空间的改进型穴度算法 ICDA (Improved Caving Degree Algorithm).对于 7 组共 700 个著名的装箱实例[1]的计算结果表明,改进后的穴度算法同时具有求解质量高和计算时间短的优点.特别是对于其中的 2 组共 200 个算例,新算法得到了高于前人的平均填充率,将 Parreño 等人于 2010 年最新发布的最好纪录分别提高了 0.12 和 0.10 个百分点.

2 基于动作空间的改进型穴度算法

改进的穴度算法包括构造型算法和邻域搜索算法两部分.其中构造型算法每一步将形状大小相同的长方体捆绑成一个大的长方体块,并挑选一个靠近箱子边缘的体积较大的动作空间来放入该捆绑块.而邻域搜索算法则是在构造型算法的基础上加以回溯处理.下面依次给出动作空间的定义、动作空间待放角区的选择方法、动作空间的选择方法以及放入到动作空间中的捆绑块的生成和选择方法,最后给出求解三维装箱问题的构造型算法和邻域搜索算法.

2.1 动作空间

设在当前格局下,箱子中已经放入了 0 至多个长方体.我们用若干个动作空间来描述当前格局下剩余的空闲空间.

定义 1. 动作空间.在当前格局下,若往箱子中合法地放入一个虚拟的长方体,该长方体的上、下、左、右、前、后 6 个面均与已放入的物体或箱子的壁相贴(即重合的面积大于 0),则该虚拟长方体所占的空间称为当前格局下的一个动作空间.

图 1 所示为在箱子的左下后角放入一个物体后当前格局下的 3 个动作空间,它们分别在该物体的右方、前方和上方.

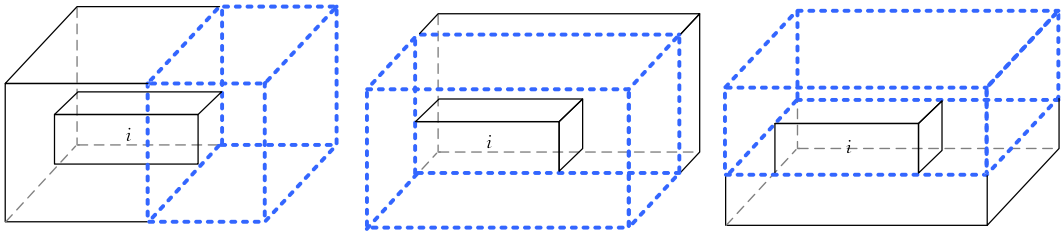


图 1 动作空间

可见,动作空间代表了当前剩余空间中可能放入的每个最大长方体的形状和位置.动作空间之间可以有重叠.而且,在动作空间中放入物体,只要不超出该空间,就可以保证该物体不会与其他已放入的物体或箱子的壁重叠.

2.2 动作空间的待放角区

每个动作空间均为三维的矩形空间,因此有8个角区.将捆绑块放入到一个动作空间时,我们选择8个角区中最靠近箱子边缘的一个角区来放入该块.具体做法是:动作空间的每个角区与箱子有一个对应角,如图2所示.分别计算动作空间的每个角区与箱子对应角在 x、y、z 方向的距离,并从小到大排列为三元组,称为该角区的角距离.然后依字典序比较不同角区角距离的大小,并选择角距离最小的一个角区作为该空间的待放角区.

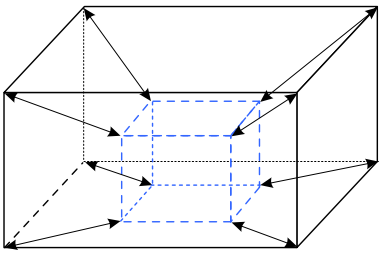


图 2 动作空间的角区与其箱子的对应角

算法对动作空间的8个角区编号,并依次考查每一个角区.下面给出选择动作空间待放角区的子算法.

算法 1. SELECT_CANDIDATE_CORNER.

输入:动作空间 *space*

输出:该空间的 *candidateCorner*

1. Begin
2. 待放角区 *candidateC*←NULL(初值为空);
3. FOR i=1 to 8:
4. 对动作空间 *space* 的每一个角区 C_i,
5. 计算该角区的角距离;
6. IF *candidateC*=null,THEN
7. *candidateC*←C_i;
8. ELSE
9. 依字典序比较 *candidateC* 与 C_i 的角距离;
10. IF C_i 的角距离较小,THEN
11. *candidateC*←C_i;
12. END IF
13. END IF
14. END FOR
15. *space.candidateCorner*←*candidateC*;
16. END SELECT_CANDIDATE_CORNER.

2.3 动作空间的选择

每个动作空间都有一个待放角区.当动作空间选定后,物体待放的角区也就随之而定.我们总是选择待放角区最靠近箱子边缘的一个动作空间来放入物体.下面给出挑选动作空间的子算法,其中传入参数为当前格局下的动作空间列表.

算法 2. SELECT_ACTION_SPACE.

输入:*List actionSpaces*

输出:*candidateS*

1. Begin
2. 候选空间 *candidateS*←NULL(初值为空);
3. FOR i=1 to *actionSpaces*.size:
4. 对 *actionSpaces* 中的每一个动作空间 S_i:
5. IF *candidateS*=NULL, THEN
6. *candidateS*←S_i;
7. ELSE 依字典序比较 *candidateS* 与 S_i 的以下参数:
   ① 动作空间候选角之角距离:小优先;
   ② 动作空间之体积:大优先;
   ③ 动作空间左前下角之 x、y、z 坐标:小优先;
   ④ 动作空间右后上角之 x、y 坐标:小优先.
8. IF S_i 优于 *candidateS*, THEN
9. *candidateS*←S_i;
10. END IF
11. END IF
12. END FOR
13. RETURN *candidateS*;
14. END SELECT_ACTION_SPACE.

2.4 捆绑块的生成

对于给定的一个动作空间,可根据当前格局下箱子外剩余长方体的尺寸、数量和可行的放置方向,由程序自动捆绑生成各种不超出该动作空间的长方体块并放入到其待放角区.将一个捆绑块放入到动

作空间待放角区的动作称为一个占角动作.

捆绑块的具体生成方法是：对于形状大小不同的每类长方体和该类长方体的每种可行的放置方向，按 xyz、yxz、xzy、zxy、yzx 和 zyx 这 6 种方式生成大的长方体块. 其中 xyz 是指按当前的放置方向放入长方体，首先尽量填满 x 方向，再尽量填满 y 方向，然后 z 方向放一层得到一个长方体块、z 方向放二层得到一个长方体块、…、最后 z 方向尽量填满得到一个长方体块；即根据此类长方体的数量，在 x、y 方向尽量填满，然后 z 方向填入一层至多层以得到不同的捆绑块. 其他捆绑方案依此类推.

图 3 所示分别为在当前动作空间中放入的 3 个捆绑块. 设当前选定了箱子左半部的动作空间且其待放角为左后下角，又设形状大小相同的某类待放长方体有 25 个. 若以竖立的方向放入到此动作空间中，按照 xzy 的捆绑方式，则可以在 x 方向捆绑 4 个、z 方向捆绑 2 个形成一个长方体组件，然后将此组件在 y 方向分别捆绑 1 层、2 层或 3 层得到 3 个不同的捆绑块. 虽然在 y 方向可以放入 4 层，但由于此类长方体只有 25 个，而放入 3 层后已经使用了 24 个，所以无法在 y 方向放入 4 层以构成新的捆绑块.

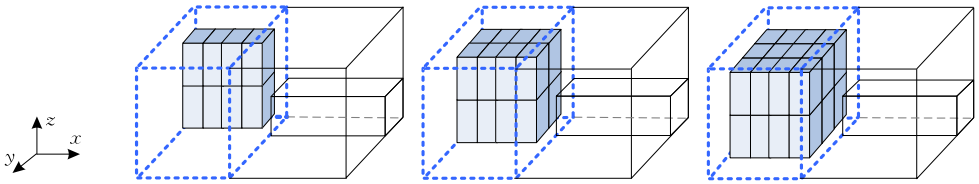


图 3　捆绑块的生成

2.5　捆绑块的选择

由于构成捆绑块的长方体类型、放置方向和捆绑方式的不同，在当前格局下可能有多个不同的放入动作. 我们定义穴度来评价不同的放入动作的优劣，并从中选择一个穴度最大的放入动作来做. 穴度越大，说明捆绑块与当前的动作空间吻合得越好. 本文将整个捆绑块视为一个物体，给出了 3 种穴度的定义.

定义 2.　贴面数 k_i. 捆绑块有多少个面与动作空间相贴即重合的面积大于 0.

定义 3.　贴面率 r_i. 捆绑块与动作空间相贴的面积与捆绑块的总表面积之比.

定义 4.　它贴面数 p_i. 其他已放入的块及箱子的壁有多少个面与捆绑块相贴.

定义 5.　邻近度 ad_i^1. 由捆绑块体积和动作空间体积的比率以及组成块的小长方体的数目即捆绑数 n_i 决定：

$$ad_i^1=\frac{1}{\sqrt[3]{n_i}}\cdot\frac{L_iW_iH_i}{L_DW_DH_D},\ ad^1\in(0,1].$$

其中 L_i、W_i、H_i 为捆绑块的三边边长，L_D、W_D、H_D 为动作空间的三边边长. 动作空间的体积利用率越大，ad_i^1 越大；组成块的捆绑数越小，ad_i^1 越大. 也称 ad_i^1 为动作空间的平均填充率.

定义 6.　邻近度 ad_i^2. 由捆绑块与动作空间未贴面的最小距离 d_i 和捆绑块三边边长的几何平均值决定：

$$ad_i^2=\exp\left(\frac{-d_i}{\sqrt[3]{L_iW_iH_i}}\right),\ ad^2\in(0,1].$$

定义 7.　邻近度 ad_i^3. 由捆绑块与动作空间未贴面的最小距离 d_i 和捆绑块三边边长的算术平均值决定：

$$ad_i^3=\exp\left(\frac{-d_i}{\frac{1}{3}(L_i+W_i+H_i)}\right),\ ad^3\in(0,1].$$

定义 8.　穴度 C_i^k. 穴度依次由贴面数、它贴面数、邻近度和贴面率决定. 根据不同的邻近度定义，相应地有三种穴度的定义.

$$C_i^k=\langle k_i,p_i,ad_i^k,r_i\rangle,\ k\in\{1,2,3\}.$$

穴度的比较即依字典序比较此穴度四元组的大小. 穴度越大，说明放入块与动作空间相贴的面越多，贴住的其他放入块或箱子的壁越多，对动作空间占满的程度越大，被贴的面积比率越大，从而与动作空间的吻合程度越好.

下面给出捆绑块的选择子算法. 根据不同的穴度定义，分别有 3 个子算法 SELECT_BLOCK^k($k\in\{1,2,3\}$). 其中的传入参数为待放的动作空间和当前格局下箱子外剩余的不同类型长方体的尺寸和数量.

算法 3.　SELECT_BLOCK^k.

输入：*ActionSpace S*，*List itemSets*

输出：*candidateB*

1. Begin
2. 捆绑块列表 *blockList*←null；
3. FOR *itemSets* 中的每种长方体类型和数量：
4. 　FOR 此类长方体的每种合法的放置方向：
5. 　　按 xyz、yxz、xzy、zxy、yzx 和 zyx 这 6 种捆绑

方式生成各种捆绑块并加入到 *blockList* 中；
6. END FOR
7. END FOR
8. 去掉 *blockList* 中形状大小完全相同的捆绑块；
9. 选择的捆绑块 *candidateB*←NULL；
10. FOR *blockList* 中的每个捆绑块 B_i：
11. IF *candidateB*=NULL，THEN
12. *candidateB*←B_i；
13. ELSE 依字典序比较*candidateB*与B_i的以下参数：
① 穴度 C_i^k：大优先；
② 构成块的小长方体之体积：大优先；
③ 构成块的小长方体之长边长、短边长：大优先；
④ 捆绑块左前下角之 x、y、z 坐标：小优先；
⑤ 捆绑块右后上角之 x、y、z 坐标：小优先；
⑥ 构成块的小长方体之方向数：小优先.
14. IF B_i优于 *candidateB*，THEN
15. *candidateB*←B_i；
16. END IF
17. END IF
18. END FOR
19. RETURN *candidateB*；
20. END SELECT_BLOCK^k.

2.6 基于动作空间的构造型穴度算法

在当前格局下，每一步选择一个最靠近箱子边缘的动作空间(待放角区随之而定)，再选择一个与当前空间吻合程度最好的捆绑块放入，然后更新动作空间列表，包括去掉已用的动作空间，加入新生成的动作空间以及更新与放入块有重叠的动作空间，从而得到一个新的格局. 如此不断迭代，直至终止格局. 根据 3 种穴度的定义，相应地有 3 种构造型算法 CONSTRUCT_SOLUTION^k ($k\in\{1,2,3\}$)，其传入参数为当前格局下的动作空间列表和箱子外剩余的不种类型长方体的尺寸和数量.

算法 4. CONSTRUCT_SOLUTION^k.

输入：*List actionSpaces*，*List itemSets*
输出：箱子的填充率
1. 当前动作空间 S←NULL；
2. WHILE *actionSpaces* 不为 NULL
3. 选择一个动作空间：
S←SELECT_ACTION_SPACE (*actionSpaces*)；
4. 为空间 S 选择一个捆绑块：
candidateB←SELECT_BLOCK^k(S,*itemSets*)；
5. IF *candidateB*=NULL，THEN
从 *actionSpaces* 中删除此空间；
6. ELSE
7. 将 *candidateB* 放入 S 的待放角区；
8. 更新 *actionSpaces* 并计算更新空间的候选角；
9. 更新箱子外的剩余长方体集 *itemSets*；
10. END IF
11. END WHILE
12. 计算并返回箱子的填充率；
13. END CONSTRUCT_SOLUTION^k.

2.7 基于动作空间的树搜索算法

在构造型算法的基础上，加入带回溯的树搜索方法，从而得到最终的确定性求解算法. 下面首先给出动作的价值度定义.

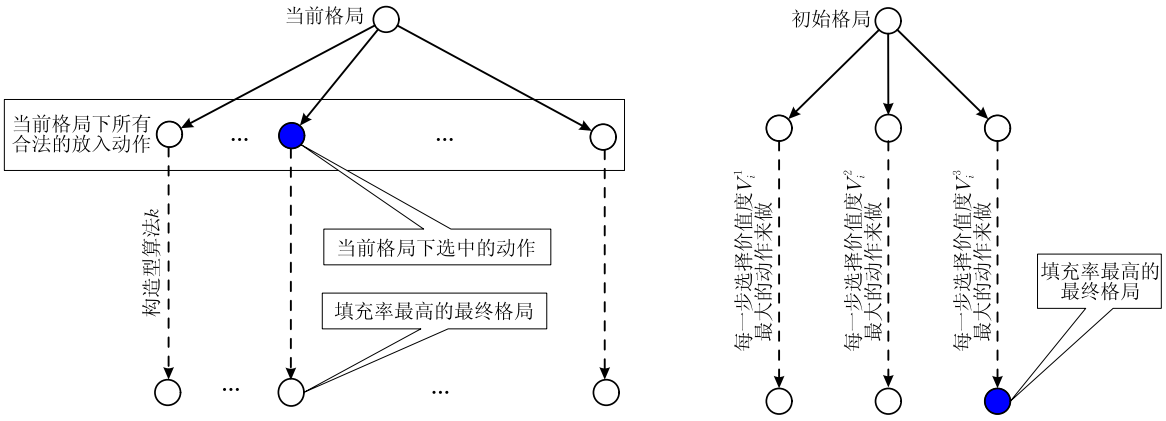
定义 9. 价值度 V_i^k. 在当前格局下作一个占角动作，得到一个新的格局. 从新格局开始，执行构造型算法 CONSTRUCT_SOLUTION^k，算法终止时箱子的填充率称为该动作的价值度.

本文分别用 3 种构造型算法进行回溯，从而得到 3 种穴度定义下箱子的填充率，然后从中选择填充率最高的一个布局作为最终的结果输出.

算法 5. TREE_SEARCH.

输入：*Container c*，*List itemSets*
输出：最终布局
1. Begin
2. 初始化：生成唯一的一个动作空间S_0并选定其候选角；
3. 空间列表 *actionSpaces*←S_0；
4. FOR k=1 to 3：
5. *actionSpaces*^{k}←复制 *actionSpaces*；
6. *itemSets*^{k}←复制 *itemSets*；
7. WHILE *actionSpaces*^{k}不为 NULL：
8. 选择一个动作空间：
S^k←SELECT_ACTION_SPACE
(*actionSpaces*^{k})；
9. 生成此空间可放入的各种捆绑块 *blockList*^{k}；
10. 计算这些捆绑块放入后的穴度 C_i^k，
并按 CONSTRUCT_SOLUTION^{k}的规则排序；
11. 记当前格局为 F^k；
12. 价值度 V_{best}←0；
13. FOR *blockList*^{k}中的每一个捆绑块 B^k：
14. 从格局 F^k开始，将 B^k放入 S^k的候选角区，
并计算此动作的价值度 V_i^k；
15. IF $V_i^k>V_{\text{best}}$，THEN V_{best}←V_i^k；
16. END FOR
17. 在格局F^k下将V_{best}对应的块放入S^k的候选角区；
18. 更新 *actionSpaces*^{k}并计算更新空间的候选角区；
19. 更新箱子外的剩余长方体集 *itemSets*^{k}；
20. F^k←新格局；
21. END WHILE
22. *result*^{k}←按价值度 V_i^k计算得到的最终布局；
23. END FOR
24. 从 3 个最终布局中选择填充率最大的一个输出；
25. END TREE_SEARCH.

树搜索算法的执行过程如图 4 所示. 其中图 4(a)所示为内层循环中在当前格局下用回溯的方式计算价值度 V_i^k 从而得到一个新的格局. 图 4(b)所示为外层循环中用 3 种价值度分别计算得到不同的最终格局，并从中选择填充率最高的作为最终的计算结果输出.



(a) 内层循环中在当前格局下用回溯方式计算　　(b) 外层循环中用3种价值度计算

图 4　树搜索算法

3　实验计算

我们将基于动作空间的改进型穴度算法 ICDA 用 Java 语言实现，在 Intel(R) Xeon(R) CPU 2.33GHz 的计算机上进行了实验计算，并与国内外代表性求解算法的计算结果进行了比较.

3.1　测试算例

实验采用的测试数据为 Bischoff 和 Ratcliff 于 1995 年提出的 7 组共 700 个弱异构型算例[1]，依次称为 BR1，BR2，…，BR7. 这 7 组测试数据为国内外引用得最多的算例集. 其中每个算例待放长方体的数目约为 130 个，不同尺寸的长方体类型从 3 种(BR1)到 20 种(BR7)不等，待放长方体的总体积接近于箱子的体积，其最优解未知.

3.2　计算结果

对这 700 个著名的三维装箱算例，国内外许多的研究者进行了实验计算. 作为比较，本文包括了其中代表性的算法 H_BR[1]、H_B_*al*[2]、GA_GB[15]、TS_BG[10]、HGA_BG[3]、PGA_GB[4]、PTS_B_*al*[12]、PH_M_*al*[5]、MFB[6]、GRASP[7]、H_B[8]、CH[9]、GRASP′[13]、HSA[11]和 VNS[14]. 表 1 给出了这些算法与改进型穴度算法 ICDA 的计算结果. 其中包括每组算例的平均填充率(%)以及全体算例的平均填充率(%). 表中标出了每组平均填充率最高的前 3 个数据，且平均填充率最高的数据用粗体显示.

表 1　不同算法的计算精度比较　　(单位：%)

算法(发表年)		算例集(类型数)							平均
		BR1(3)	BR2(5)	BR3(8)	BR4(10)	BR5(12)	BR6(15)	BR7(20)	
H_BR	(1995)	83.37	83.57	83.59	84.16	83.89	82.92	82.14	83.37
H_B_*al*	(1995)	81.76	81.70	82.98	82.60	82.76	81.50	80.51	81.97
GA_GB	(1997)	86.77	88.12	88.87	88.68	88.78	88.53	88.36	88.30
TS_BG	(1998)	92.63	92.70	92.31	91.62	90.86	90.04	88.63	91.26
HGA_BG	(2001)	87.81	89.40	90.48	90.63	90.73	90.72	90.65	90.06
PGA_GB	(2002)	88.10	89.56	90.77	91.03	91.23	91.28	91.04	90.43
PTS_B_*al*	(2003)	93.52	93.77	93.58	93.05	92.34	91.72	90.55	92.65
PH_M_*al*	(2004)	93.70 (3)	94.30 (3)	94.54 (2)	94.27 (3)	93.83 (3)	93.34 (3)	92.50 (2)	93.78 (3)
MFB	(2005)	87.40	88.70	89.30	89.70	89.70	89.70	89.40	89.10
GRASP	(2005)	89.07	90.43	90.86	90.42	89.57	89.71	88.05	89.73
H_B	(2006)	89.39	90.26	91.08	90.90	91.05	90.70	90.44	90.55
CH	(2007)	89.94	91.13	92.09	91.94	91.72	91.45	90.94	91.32
GRASP′	(2009)	93.27	93.38	93.39	93.16	92.89	92.62	91.86	92.94
HSA	(2009)	93.81	93.94	93.86	93.57	93.22	92.72	91.99	93.30
VNS	(2010)	**94.93 (1)**	**95.19 (1)**	**94.99 (1)**	**94.71 (1)**	**94.33 (1)**	94.04 (2)	93.53 (2)	**94.53 (1)**
ICDA	(2010)	93.98 (2)	94.59 (2)	94.52 (3)	94.41 (2)	94.23 (2)	**94.16 (1)**	**93.63 (1)**	94.22 (2)

可见,在这 16 个算法中,平均填充率最高的前 3 个算法依次为 Parreño 等人于 2010 年最新发布的 VNS 算法、我们的 ICDA 算法和 Mack 等人于 2004 年提出的 PH_M_*al* 算法.其中 ICDA 算法对每组算例的平均填充率的排名依次为 2、2、3、2、2、1 和 1,且在 BR6 和 BR7 这两组算例上的填充率要高于目前已见发表的最好结果.

一般而言,随着算例异构性的增强,即长方体的类型越来越多,同类长方体的数量越来越少时,由于长方体对齐的难度增加而不可避免地产生各种空隙,因此算法求解的优度呈下降的趋势.对于 ICDA 算法,在待放长方体的类型较多同时每类长方体的数目也较多的情况下,可生成的不同捆绑块的种类较多,从而有可能找到与当前动作空间吻合程度高的块放入,因此在一定程度上缓解了算例异构性增强带来的不利影响.从表 1 中可看出,VNS 算法在最好结果与最坏结果之间的差距为 1.66,而 ICDA 算法的仅为 0.96.这使得 ICDA 算法在 BR6 和 BR7 这两组算例上得到了好于 VNS 的结果.

表 2 给出了其中的若干个算法实验计算时所用的机器配置和运行的时间.ICDA 算法在全体算例上的平均计算时间为 53.90 s.可见,ICDA 算法的计算精度较高,同时计算的时间很短,是一个高效率的确定性快速求解算法.

表 2 若干算法的计算速度比较

算法	CPU 配置/GHz	运行时间/s
PGA_GB	5×0.4	183.00
PTS_B_*al*	4×2.0	121.00
PH_M_*al*	4×2.0	222.00
GRASP	2.40	33.71
CH	2.60	85.08
GRASP′	1.50	8.00
HSA	2.00	65.87
VNS	1.50	28.00
ICDA	2.33	53.90

4 结 论

本文基于当前可能放入的最大虚拟长方体所对应的动作空间重新定义了穴度,并在放入位置的选择上直接体现"金角银边草肚皮"的思想,使新算法克服了原有穴度算法计算时间较长的弱点.对 700 个 BR 算例的计算结果表明,新算法的空间填充率很高,同时计算的速度很快.由于 NP 难度问题不存在多项式时间的确定性精确求解算法,除非 P=NP.因此,本文提出的求解三维装箱这一 NP 难度问题的快速高效的确定性算法具有较高的理论与实际价值.

参 考 文 献

[1] Bischoff E E, Ratcliff M S W. Issues in the development of approaches to container loading. OMEGA—The International Journal of Management Science, 1995, 23(4): 377-390

[2] Bischoff E E, Janetz F, Ratcliff M S W. Loading pallets with non-identical items. European Journal of Operational Research, 1995, 84(3): 681-692

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

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

[5] Mack D, Bortfeldt A, Gehring H. A parallel hybrid local search algorithm for the container loading problem. International Transactions in Operational Research, 2004, 11(5): 511-533

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

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

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

[9] Zhang De-Fu, Wei Li-Jun, Chen Qing-Shan, et al. 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,陈青山等. 三维装箱问题的组合启发式算法. 软件学报, 2007, 18(9): 2083-2089)

[10] Bortfeldt A, Gehring H. Applying tabu search to container loading problems//Kall P, Lüthi H J eds. Operations Research Proceedings. Zurich: Springer, 1998: 533-538

[11] Zhang De-Fu, Peng Yu, Zhu Wen-Xing, et al. A hybrid simulated algorithm for the three-dimensional packing problem. Chinese Journal of Computers, 2009, 32(11): 2147-2156(in Chinese)
(张德富,彭煜,朱文兴等. 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)

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

[13] Parreño F, Alvarez-Valdes R, Tamarit J M, et al. A maximal-space algorithm for the container loading problem. INFORMS Journal on Computing, 2008, 20(3): 412-422

[14] Parreño F, Alvarez-Valdes R, Oliveira J F, et al. Neighborhood structures for the container loading problem: A VNS implementation. Journal of Heuristics, 2010, 16(1): 1-22

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

[16] Huang Wen-Qi, He Kun. A caving degree approach for the single container loading problem. European Journal of Operational Research, 2009, 196(7): 93-101

[17] Huang Wen-Qi, Liu Jing-Fa. A deterministic heuristic algorithm based on Euclidian distance for solving the rectangles Packing problem. Chinese Journal of Computers, 2006, 29(5): 734-739(in Chinese)
(黄文奇，刘景发. 基于欧氏距离的矩形 Packing 问题的确定性启发式求解算法. 计算机学报，2006，29(5)：734-739)

HE Kun, born in 1972, Ph. D., associate professor. Her research interests include algorithms for NP hard problems and data mining.

HUANG Wen-Qi, born in 1938, professor, Ph. D. supervisor. His research interest is algorithms for NP hard problems.

Background

This work is supported by the National Natural Science Foundation of China under Grant No. 61173180. The project mainly focuses on efficient methods for solving the container loading problem and the equal circle packing problem. This paper belongs to one of the key parts of the project, and focuses on efficient quasi-human algorithms for solving the container loading problem. This three-dimensional packing problem is NP hard and finds many applications in the transportation and computer industry. Various efficient heuristics have been presented in the world, including wall building, cuboid arrangement, caving degree and maximal space. In this paper, based on the caving degree approach that the authors proposed previously, the authors present a new efficient algorithm that not only increases the volume utilizations markedly but also shorts the running times evidently.