求解三维装箱问题的多层启发式搜索算法

张德富[1)] 彭 煜[2)] 张丽丽[1)]

[1)](厦门大学信息科学与技术学院 福建 厦门 361005)

[2)](香港大学计算机科学系 香港)

摘 要 文中提出了一个高效求解三维装箱问题的多层启发式搜索算法.该算法基于块装载的思想,按照块选择算法确定每个阶段采用的块,然后以一种固定的装载方式装载块,直到无法继续装载.文中的主要贡献在于发展了一个有效的复合块生成算法,特别的,提出了基于多层搜索的块选择算法,该算法用多层搜索来评价可行块,然后选择最合适的块进行装载.对1500个三维装箱问题测试数据的计算结果表明,提出的算法几乎在所有测试数据上的填充率都超过了目前已知的优秀算法.

关键词 三维装箱问题;启发式算法;深度优先搜索

中图法分类号 TP301 **DOI号**: 10.3724/SP.J.1016.2012.02553

A Multi-Layer Heuristic Search Algorithm for Three Dimensional Container Loading Problem

ZHANG De-Fu[1)] PENG Yu[2)] ZHANG Li-Li[1)]

[1)](*School of Information Science and Technology, Xiamen University, Xiamen, Fujian* 361005)

[2)](*Department of Computer Science, University of Hong Kong, Hong Kong*)

Abstract This paper presents an efficient multi-layer heuristic search algorithm for three-dimensional container loading problem. This algorithm loads one block, which is determined by block selection algorithm, in one packing phase according to a fixed strategy until no blocks are available. Two algorithms are developed, one is used to efficiently generate complex block, another is block selection algorithm based on multi-layer search that is used to evaluate the blocks so that the appropriate block is selected. Computational results on 1500 test data of three-dimensional container problem show that the proposed algorithm outperforms the best known algorithm in almost all the test data.

Keywords three-dimensional container loading problem; heuristic algorithm; depth-first search

1 引 言

在物流、运输等工业领域,经常会遇到三维装箱问题.如何提高装箱的效率,已经成为科学研究和实践中非常关注的一个重要课题.求解装箱问题的高效求解算法,对降低成本、提高企业的赢利能力,有着非常积极的现实意义.同时,三维装箱问题又是一个NP难题,虽然研究的历史很长,但是很难获得有效的求解算法.

由于装箱问题出现在不同的应用领域,实际应用中的装箱问题有不同的优化目标和装载约束,从而派生出许多问题.Dyckhoff等人[1]总结了不同类型的装箱问题,并给出了相应的分类,例如单容器装

收稿日期:2010-05-12;最终修改稿收到日期:2011-08-25.本课题得到国家自然科学基金(61272003)资助.张德富,男,1971年生,博士,教授,主要研究领域为计算智能、金融数据挖掘.E-mail:dfzhang@xmu.edu.cn.彭 煜(通信作者),男,1984年生,博士研究生,主要研究方向为算法、计算智能.E-mail:ypeng@cs.hku.hk.张丽丽,女,1974年生,博士研究生,主要研究方向为多目标优化.

箱问题和多容器装箱问题;同构装箱问题(只含一种类型的箱子)和异构装箱问题(包含多种类型的箱子).本文研究单容器装箱问题,其形式化的定义可以描述如下:

给定一个容器(其体积为 V)和一系列待装载的箱子,容器和箱子的形状都是长方体.问题的目标是要确定一个可行的箱子装载方案,使得在满足给定装载约束的条件下,使容器中所装箱子总体积 S(或者填充率 $S/V\times100\%$)尽可能的大.可行装载方案要求必须满足如下两个条件:

(1)任一装载的箱子不能与其它装载箱子或者容器互相重叠.

(2)所有装载的箱子以与容器平行的方式装载.

此外,根据实际问题的需要,本文还考虑如下约束条件:

(C1)方向性约束.在许多应用中,箱子的装载有方向性约束.也就是说,每个箱子只有它的 1 条或 2 条边可以竖直放置作为高度,即箱子的某个面必须朝上.

(C2)稳定性约束.在实际应用中,例如物流领域,装载必须满足稳定性约束.这意味着每个被装载的箱子必须得到容器底部或者其它已经装载箱子的支撑.根据实际应用需要,有完全支撑约束,即被装载箱子的底部必须跟其它已经装载箱子完全接触,不允许底部有悬空的部分,部分支撑约束指的是装载箱子的底部,可以允许部分悬空.

由于三维装箱问题是一个典型的 NP 难题[2],因此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.虽然一些研究采用了精确算法,但其求解的规模有限,因此启发式求解方法成为理论研究和实际应用的首选.

对于三维装箱问题,基于垂直"层"或"墙"概念的启发式算法比较多.George 等人[3]首先提出了基于层的启发式方法.Bischoff 等人[4]比较了 14 种基于层的方法.Bortfeldt 等人[5]在层概念的基础上,设计了一种混合遗传算法.Pisinger[6]基于层的概念,将整个容器空间分成若干垂直的层,再将层分成若干水平或垂直的条形,然后利用背包问题的算法来求解.Bischoff 等人[7]针对异构装箱问题,以自底向上的摆放思想,提出了基于平面的算法.Gehring 等人[8]基于塔的概念,设计了一个遗传算法.Bortfeldt 等人[9]提出了一个禁忌搜索算法.与"塔"和"层"的概念不同,Eley[10]设计了基于同类块(Block)的算法.Bortfeldt 等人[11]进一步拓展了"块"的概念,然后使用禁忌搜索寻找最优的装载序列作为问题的近似解.Moura 等人[12]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).Parreño 等人[13]进一步发展和改进了该算法,取得了不错的结果.Parreño 等人[14]基于最大空间的概念,提出了一个可变邻域搜索算法.Fanslau 等人[15]基于"块"的概念,提出了"复合块"的思想,然后设计了一个有效的启发式树状搜索算法,是目前解决装箱问题最有效的算法.此外 Ngoi、Bischoff、Morabito、Sixt、Gehring、Lim、Juraitis 和 Bortfeldt 等人也报告了其它一些有趣的启发式算法,并将他们应用到三维装箱问题中[16-24].国内学者对三维装箱问题的研究,也取得了一些不错的结果[25-31],例如张德富等人[29]提出了一个混合模拟退火算法,Huang 等人[31]提出了一个有效的拟人型穴度算法.

本文在文献[15,29]研究的基础上,提出了一个多层启发式搜索算法,这个算法与文献[15]不同的地方在于,文献[15]采用基于整数拆分的树状搜索算法,而本文采用深度+宽度搜索的思想,提出多层搜索算法,能更有效地评价可行块以便选择一个近似最优的块进行装载.与文献[29]最大的不同在于文献[29]从一个初始装载序列出发,采用模拟退火算法来搜索一个好的装载序列,作为问题的近似解,而本文采用构造性算法,用多层搜索思想来选择一个近似最优块进行装载,然后逐步构造直到获得一个装载序列.实验结果验证了提出算法的有效性,而且本文算法超过了已出版的优秀算法.

2 多层启发式搜索算法

2.1 基本的块装载启发式算法

因为装箱问题本身的复杂性和实际应用的需要,必须使用启发式方法才能有效地生成装载方案.一个优秀的启发式算法不但应该能迅速找到解,而且应该尽可能地找到接近最优的解,同时算法还应该展示必要的灵活性使其能够适应不同的场合和需求.本文提出的基于块装载的三维装箱问题的求解算法,是受日常装箱经验的启发而提出的,一些算法思想与文献[11,29]中的类似.

算法 1 给出了基本的块装载启发式算法的具体描述,算法首先根据输入参数指定的 $isComplex$ 生成所有可能的简单块或复合块;接着初始化当前部分装载方案,并开始装载过程;每个装载阶段,算法从剩余堆栈栈顶取出一个空间 $space$,用 GenBlock-

List 算法生成它的可行块列表；当列表不为空时，FindNextBlock 算法选择一个块进行装载并加入当前部分装载方案，接着采用 GenResidulSpace 划分未填充的空间并将它们插入堆栈；当列表为空时，算法 TransferSpace 尝试将 *space* 中可利用的部分转移到堆栈中相应的剩余空间.

算法 1.　块装载启发式算法.

```
BasicHeuristic(isComplex, searchParams, problem)
if isComplex then
  blockTable := GenComplexBlock(problem.container,
                problem.boxList, problem.num)
else
  blockTable := GenSimpleBlock(problem.container,
                problem.boxList, problem.num)
endif
set search parameters according to searchParams
ps.avail := problem.num
ps.plan := {}
ps.volume=0
ps.spaceStack := {}
ps.spaceStack.push(problem.container)
while ps.spaceStack ≠ {} do
  space := ps.spaceStack.top()
  blockList := GenBlockList(ps.space, ps.avail)
  if blockList≠{} then
    block := FindNextBlock(ps,blockList)
    ps.spaceStack.pop()
    ps.avail := ps.avail−block.require
    ps.plan := ps.plan+(space,block)
    ps.plan.volume := ps.plan.volume+block.volume
    ps.spaceStack.push(GenResidulSpace(space,block))
  else
    TransferSpace(space, ps.spaceStack)
  endif
endwhile
return ps.plan
```

其中，*space* 表示剩余空间，采用参考点加 3 边长度的方法来表示；*problem* 包含容器、*box* 列表以及可用箱子向量，形式化表示了一个装箱问题；*block* 结构表示块，可以是简单块或复合块，它有一个 *require* 向量指出其包含的所有箱子的数量. 关于算法采用的数据结构以及有关变量的含义，可以参看文献[29]. 下面，本文提出的多层启发式算法，以块装载启发式算法为基础，嵌入了下面两个核心算法.

2.2　简单块、复合块及其生成算法

简单块是由同一朝向的同种类型的箱子堆叠而成的，箱子和箱子之间没有空隙，堆叠的结果必须恰好形成一个长方体. GenSimpleBlock 算法[29]枚举所有可行的组合(nx,ny,nz)，并将其对应的简单块加入块表. 其中可行组合应满足：所包含的箱子数目小于对应可用箱子数目，且块大小应小于容器大小.

复合块是通过不断组合简单块而得到的，其定义[29]如下：

(1) 简单块是最基本的复合块.

(2) 给定两个复合块 a,b. 可以按 3 种方式进行组合得到复合块 c：按 x 轴方向组合，按 y 轴方向组合，按 z 轴方向组合. c 是包含 a、b 的最小长方体.

显然，按照上述定义，复合块的数量将是箱子数目的指数级，而且任意组合生成的复合块中可能有很多空隙，非常不利于装载. 因此，有必要对复合块施加一定的限制，本文的限制条件基本上同文献[29]，但是本文还考虑下列条件：

如果考虑约束 C2，受复合块中空隙的影响，复合块顶部有支撑的可放置矩形可能很小，为了保证在后面的装载过程中剩余空间不会由于失去支撑而浪费可用空间，我们限定顶部可放置矩形与相应的复合块顶部面积的比至少要达到 *MinAreaRate*.

上述条件是文献[15,29]中没有考虑的，它能减少对块的搜索，加快计算的速度.

在满足以上约束的情况下，块数目仍然可能很大，生成算法将在块数目达到 *MaxBlocks* 时停止生成.

根据复合块的定义，可以得到最终的复合块生成算法 2. 此算法首先调用简单块生成算法生成所有可能的简单块；接着，迭代 *MaxTimes* 次，在每一次迭代中，对于任何两个已生成的复合块 a、b，尝试按 x 轴、y 轴、z 轴方向进行组合；如果组合满足前面所述的组合限制条件，则将新生成的复合块 c 加入块列表.

算法 2.　复合块生成算法.

```
GenComplexBlock(container, boxList, num)
blockTable := GenSimpleBlock(space, boxList, num)
for times := 0 to MaxTimes−1 do
  newBlockTable := {}
for each a, b in blockTable do
  if a.times=times or b.times=times then
    if a and b satisfied the constraints in x-direction
    then
      c := combine a, b in x-direction
      add c to newBlockTable
    endif
    if a and b satisfied the constraints in y-direction
    then
      c := combine a, b in y-direction
```

```
        add c to newBlockTable
      endif
      if a and b satisfied the constraints in z-direction
      then
        c := combine a,b in z-direction
        add c to newBlockTable
      endif
    endif
  endfor
  blockTable := blockTable + newBlockTable
  reduce duplicated block in blockTable
  endfor
  sort blockTable by decreasing volume of blocks.
  return blockTable
```

介绍了块的概念后，下面描述可行块列表生成算法 GenBlockList(*space*, *avail*). 该算法用于从 *blockTable* 中获取适合当前剩余空间的可行块列表. 其中，*blockTable* 是在算法开始时预先生成的所有可能的块的列表，以避免重复计算. 这样，在某一时刻计算某一个剩余空间的可行块列表时，只要扫描 *blockTable*，找出所有能放入该剩余空间且能被当前剩余箱子满足的块即可. 这样使得基本的块装载启发式算法与块生成算法完全无关，块生成算法可以根据需要进行定制而不影响块装载启发式算法.

算法 GenBlockList(*space*, *avail*) 描述了可行块列表生成算法，该算法扫描 *blockTable*，返回所有能放入剩余空间并且有足够剩余箱子的块. 由于 *blockTable* 是按块中箱子总体积降序排列的，返回的可行块列表 *blockList* 也是按箱子总体积降序排列.

在每个装载阶段一个剩余空间被装载，装载分为有可行块和无可行块两种情况. 在有可行块时，算法按照块选择算法选择可行块，然后将未填充空间划分成新的剩余空间，这里划分的策略是使划分出的剩余空间尽可能地大. 图 1 给出了具有稳定性约束的剩余空间的划分过程. 令 x 轴、y 轴和 z 轴上的剩余长度分别为 mx、my、mz. 如果 $my \geq mx$，则按图 1(a) 划分，划分空间的入栈顺序为 *spaceZ*, *spaceX*, *spaceY*；如果 $mx \geq my$，则按图 1(b) 划分，空间的入栈顺序 *spaceZ*, *spaceY*, *spaceX*. 在无可行块时，当前剩余空间被抛弃，若其中一部分空间可以合并到当前堆栈中的其它空间，则进行空间转移，重新利用这些空间. 可转移空间可以被转移给剩余空间堆栈中来自同一次划分的其它空间以重新利用. 可转移空间的重新分配实际上就是对未填充空间的重新划分. 因此，可以通过重新划分未填充空间来达到再次利用可转移空间的目的. TransferSpace(*space*, *spaceStack*) 实现空间转移，此过程判定当前剩余空间与栈顶的一个或两个剩余空间是否是由同一次划分而产生的，若是则将可转移空间转移给相应的一个或两个剩余空间. 其中一些细节，可参考文献[29].

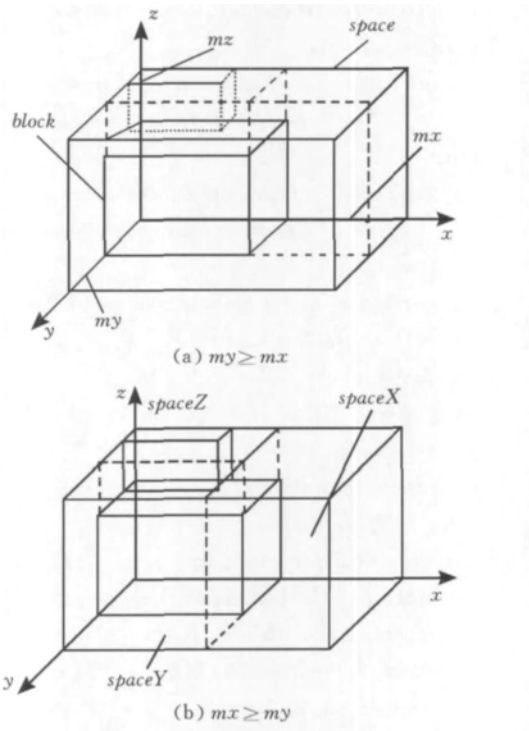

图 1 剩余空间切割

2.3 基于多层搜索的块选择算法

Fanslau 等人[15]采用如下基于整数拆分的树状搜索算法进行块选择：给定一个整数拆分 $d_0 + d_1 + \cdots + d_{n-1} = d$，搜索分为 n 层，r 为初始根节点，在第 i 层从根节点 r_i 出发进行一次深度为 d_i 的搜索，选择最优的叶节点作为新的根节点 $r_i + 1$，r_n 是最终的搜索结果. 为限制搜索的开销，每次搜索的节点数目限制为 $b_i = \max\{b \mid b^{d_i} < effort\}$. 尽管这种算法取得了相当优秀的结果. 但是，仔细分析该算法的执行过程，它仍然有一些缺陷：

(1) 在指定整数拆分下，算法每个阶段只选择当前状态下的局部最优解，并在下一阶段以此为基础进行搜索. 但是局部最优毕竟不是全局最优，一旦在某个阶段选择错误，后续的评估结果就必然会出现偏差. 虽然算法通过枚举所有的整数拆分允许搜索更多的节点，但是由于每次总是选择局部最优的那个节点，算法很难选取到那些要到很大的深度才能体现其优势的块.

(2)算法在执行搜索过程中，有很大一部分重复计算，一方面是因为不同的整数拆分方式有一部分重叠，另一方面是因为不同拆分所找到的节点很可能是相同的.

针对树状搜索算法的两个缺陷，本文提出了多层启发式搜索，这里“层”的概念对应于整数拆分里面的每一个拆分，但是每一层的深度为 1. 第 i 层的节点是初始局部装载方案装载了 i 个块以后的结果. 该算法与基于整数拆分的树状搜索算法最大的不同在于：在第 i 层，算法不再只选择一个块继续搜索，而是选择最优的 $MaxHeap$ 个局部装载方案，采用不同的深度 d 调用深度优先搜索算法，并将所有搜索结果都插入到相应第 $i+d$ 层的结果中去. 在每个层上，算法使用堆存储最优的 $MaxHeap$ 结果. 算法从 0 层开始迭代，直到 $depth$ 层结束. 整个算法的运行过程类似树状搜索，不过此时在每个层上有至多 $MaxHeap$ 个节点被扩展，并且已计算的上层节点不会被再次计算.

算法 3.　多层启发式搜索.

```
MultiLayerSearch(ps, depth, maxD, MaxHeap, effort)
result.volumeComplete := 0
add ps to heap[0]
for layer := 0 to depth−1 do
  keep heap[layer] containing only the best MaxHeap elements
  for each ps′ in heap[layer] do
    for d := 1 to maxD do
      branch := max{b | b^d ≤ effort}
      DepthFirstSearch(ps′, d, branch, layer+d)
    endfor
  endfor
endfor
return the maximum in heap[depth]
```

算法 3 描述了多层启发式搜索算法. $depth$ 参数描述了要装载的块数目；$maxD$ 指定了在每个层上将尝试的最大深度值；$MaxHeap$ 如前所述是每个层上用堆维护的最优的部分装载方案的最大数目；$branch$ 表示可放置块的最大数目，意味着当前层的宽度；$effort$ 正如前文所述用于限制每次搜索访问的节点的数目，对于每一个搜索深度 d，相应的分支数目 b 被计算出以用于搜索. 算法 4 是改进的带深度限制的深度优先搜索算法，其不只是记录局部最优解，而且将所有叶子节点的评估结果都加到相应层上. 其中 $PlaceBlock$，$RemoveBlock$ 抽象了块装载和块移除的过程.

算法 4.　改进的带深度限制的深度优先搜索算法.

```
DepthFirstSearch(ps, depth, branch, layer)
if depth ≠ 0 then
  space := ps.spaceStack.top()
  blockList := GenBlockList(ps.space, ps.avail)
  if blockList ≠ {} then
    for i := 0 to branch−1 do
      PlaceBlock(ps, blockList[i])
      DepthFirstSearch(ps, depth−1, branch, layer)
      RemoveBlock(ps, blockList[i], space)
    endfor
  else
    TransferSpace(space, ps.spaceStack)
    DepthFirstSearch(ps, depth, branch, layer)
    TransferSpaceBack(space, ps.spaceStack)
  endif
else
  Complete(ps)
  add ps to heap[layer]
endif
```

算法 5 描述了块装载算法，主要的任务是将块和栈顶空间结合成一个装载加入当前装载方案，移除栈顶空间，去掉已使用箱子，然后划分未填充空间并加入到剩余空间堆栈中. 其中 TransferSpace 用于将新生成的剩余空间加入堆栈，TransferSpaceBack 是其逆过程，用于取消被加入的剩余空间以便进行搜索. 当搜索进行到目标深度时，Complete(ps)算法每次放入体积最大的块直至获得一个完整的装载方案.

算法 5.　块装载算法.

```
PlaceBlock(ps, block)
space := ps.spaceStack.top()
ps.spaceStack.pop()
ps.avail := ps.avail−block.require
ps.plan := ps.plan+(space, block)
ps.plan.volume := ps.plan.volume+block.volume
ps.spaceStack.push(GenResidulSpace(space, block))
```

算法 6 描述了块移除算法，它是算法 5 的逆过程，完成的工作包括从当前部分装载方案中移除当前块所属的装载，恢复已使用箱子，移除空间堆栈栈顶的 3 个划分出来的剩余空间，并将已使用剩余空间重新插入栈顶.

算法 6.　块移除算法.

```
RemoveBlock(ps, block, space)
ps.avail := ps.avail+block.require
ps.plan := ps.plan−(space, block)
ps.plan.volume := ps.plan.volume − block.volume
```

remove 3 top spaces from *ps.spaceStack*
ps.spaceStack.push(*space*)

图 2(a)和(b)分别显示了多层启发式搜索算法的两个运行实例,图中只列出了最优节点相关的路径和分支,忽略了其它一些节点. 图 2(a)显示了一个典型的运行结果,算法从第 0 层开始执行深度为 1 和 2 的深度优先搜索,在 1 层中算法扩展最优的两个节点,其中一个节点以深度 2 执行另一次深度优先搜索后找到整个过程中的最优节点(见图中灰色节点). 图 2(b)描述的实例运行过程类似,区别在于这一次最优节点是在一次深度为 2 的深度优先搜索后接着一次深度为 1 的搜索中找到的.

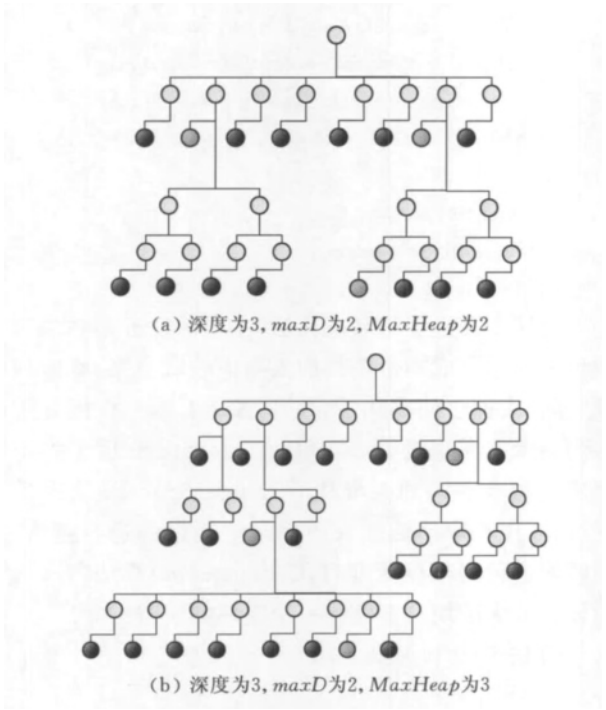

图 2 多层启发式搜索

现在利用多层搜索算法来设计块选择算法. 由于 $k+1$ 层最优的 N 个解很可能只来自 k 层最优的 N 个解,我们不需要对所有的可行块执行最大深度的多层搜索,可以用不同的深度进行搜索,每次采用贪心算法过滤掉一些不够优秀的可行块. 只要保证选择的 N 足够大,使得最终的最优解以很大概率被包含即可.

算法 7 描述了块选择算法的结构. 算法采用递增的搜索深度执行多层搜索,遍历整个可行块列表,尝试装载当前块到当前部分装载方案,然后用多层搜索来评估此部分装载方案,并将多层搜索的最优填充值作为被选块的适应度. 排序块列表中块的适应度,每次过滤掉一半不够优秀的块,直到可行块数目不大于 N. 最后返回具有最大适应度的块. 如果考虑 C2 约束,则只要判断当前块的底部是否有满足要求的支撑即可.

算法 7. 块选择算法.

```
FindNextBlock(ps,blockList)
for depth := 1 to maxDepth do
  for each block in blockList do
    space := ps.spaceStack.top()
    PlaceBlock(ps,block)
    block.fitness := MultiLayerSearch(ps,depth,
                     maxD,MaxHeap,effort)
    Remove(ps,block,space)
  endfor
  sort the blockList by decreasing fitness
  if size(blockList)>2N then
    blockList := the first half of blockList
  else
    blockList := first N in blockList
  endif
endfor
return the block with maximum fitness
```

3 实验与结果

3.1 算法参数和测试数据

本文的多层启发式搜索算法(MLHS)以 C++ 实现,实验程序运行在 Intel® Xeon® X5460 @ 3.16 GHz 处理器上. 运行环境为 Debian Linux,编译器为 gcc 4.3.2. 实验中设置的各项常数如表 1 所示.

表 1 算法常数

常数项	值	含义
MinFillRate	0.98	复合块的最低填充率
MinAreaRate	0.96	可放置矩形与相应复合块顶部面积比
MaxTimes	5	生成块的最大复杂次数
MaxBlocks	10000	最大块数目
N	16	每层最优解数目
maxD	2	每层搜索尝试的最大深度
MaxDepth	6	分层搜索的总次数
MaxHeap	6	每层扩展的节点数

MLHS 将在不同的参数设置下运行,参数按照从简单到复杂的顺序执行,并选择其中最优的结果作为最终解. 由于当前一组参数被执行完毕后,所有程序运行时间可能会超过时间限制(*Time-Limit*). 因此,当算法运行超过时间限制则在完成当前参数下的计算后,不再执行下一组参数. 实验中一共采用 6 组参数设置,如表 2 所示. 其中图 2 描述了参数 2 的参数设置,其分支 $branch=2$.

表 2　算法参数

	MaxDepth	*Effort*	*Time-Limit*
参数 1	2	1	120 s
参数 2	3	3	
参数 3	4	9	
参数 4	5	27	
参数 5	6	81	
参数 6	7	243	

本文实验采用的测试数据来自文献[17]，包括 BR1～BR15 一共 1500 个三维装箱实例，它们可以从 OR-Library① 下载或者 http://59.77.16.8/Download.aspx#p4 网站上下载. 这些实例共分为 15 种类型，每类 100 个问题. 每类问题中有一定数量的相同类型(具有相同的长宽高)的箱子，这 15 类问题中箱子的类型数从 1 到 100，异构性由弱到强，能够很好地测试算法在不同异构性装箱问题中的表现. 其中，BR1～BR7 的箱子类型数为 3～20，属于弱异构装箱问题；BR8～BR15的箱子类型数从 30～100，属于强异构装箱问题.

3.2　计算结果

对于来自文献[17]的1500个实例，许多研究者对它们或其中一部分做过测试. 比较的算法包括顺序和并行执行的禁忌搜索算法(PTSA)[11]、MFB 算法[21]、随机启发式算法[22]、H_B 算法[23]、启发式算法(SPBBL-CC4)[24]、组合启发式算法(CH)[28]. 这些算法致力于研究弱异构装箱问题，即仅测试 BR1～BR7. 特别地，本文还比较了 H_BR 算法[17]、GA_GB 算法[8]和禁忌搜索算法 TS_BG[9]、贪心随机自适应搜索算法(GRASP)[12]、maximal-space 算法[13]、可变邻域搜索算法(VNS)[14]、混合模拟退火算法(HSA)[29]、最新的基于整数拆分的树状搜索算法(CLTRS)[15]以及 FDA 算法[30]. 这些算法测试了 BR1～BR15 的 1500 个实例. 此外 Huang 等人[31]提出的 A_2 算法只计算了 BR8～BR15.

上述比较的算法均满足方向性约束 C1，有些算法也满足稳定性约束 C2，它们的计算结果均直接来自于相应文献. 表 3 与表 4 报告了这些算法与 MLHS 的计算结果，斜体数据表示问题满足约束 C1 和 C2. 表中所有的数据表示填充率(%)，即不同算法针对一类问题所得到的平均填充率，而 Mean 表示算法对所计算问题类的平均填充率.

表 3　各种算法对 BR1～BR7 的填充率比较

算法	满足约束	填充率/%							
		BR1	BR2	BR3	BR4	BR5	BR6	BR7	Mean
H_BR[17]	C1&C2	83.79	84.44	83.94	83.71	83.80	82.44	82.01	83.5
GA_GB[8]	C1&C2	85.80	87.26	88.10	88.04	87.86	87.85	87.68	87.5
TS_BG[9]	C1&C2	87.81	89.40	90.48	90.63	90.73	92.72	90.65	90.1
PTSA[11] 顺序	C1	93.23	93.27	92.86	92.40	91.61	90.86	89.65	92.0
PTSA[11] 并行	C1	93.52	93.77	93.58	93.05	92.34	91.72	90.55	92.7
MFB 算法[21]	C1	87.4	88.7	89.3	89.7	89.7	89.7	89.4	89.1
随机启发式算法[22]	C1	88.4	89.0	89.4	89.5	89.4	89.6	89.5	89.26
H_B[23]	C1	89.39	90.26	91.08	90.90	91.05	90.70	90.44	90.55
SPBBL-CC4[24]	C1	87.3	88.6	89.4	90.1	89.3	89.7	89.2	89.09
组合启发式算法 CH[28]	C1	89.94	91.13	92.09	91.94	91.72	91.45	90.94	91.32
GRASP[12]	C1	93.52	93.77	93.58	93.05	92.34	91.72	90.55	92.65
maximal-space[13]	C1	93.85	94.22	94.25	94.09	93.87	93.52	92.94	93.82
VNS[14]	C1	94.93	95.19	94.99	94.71	94.33	94.04	93.53	94.53
HSA[29]	C1&C2	93.81	93.94	93.86	93.57	93.22	92.72	91.99	93.30
CLTRS[15]	C1	95.05	95.43	95.47	95.18	95.00	94.79	94.24	95.02
	C1&C2	94.51	94.73	94.74	94.41	94.13	93.85	93.20	94.22
FDA[30]	C1	92.92	93.93	93.71	93.68	93.73	93.63	93.14	93.53
MLHS	C1	94.92	95.48	95.69	95.53	95.44	95.38	94.95	95.34
	C1&C2	94.49	94.89	95.20	94.94	94.78	94.55	93.95	94.69

从表 3 中可以看出，多层启发式算法 MLHS 几乎在每类问题上的填充率都超过了目前已知的算法，平均填充率超过了所有比较的算法. 当只考虑 C1 时，MLHS 在 BR1～BR7 上的平均填充率达到了 95.34%，比当前最新的 VNS[14]、FDA[30]、最好的 CLTRS[15]分别提高了 0.81%，1.81%和 0.32%. 当 C1 和 C2 均考虑时，比当前较新的 HSA[29]、最好的 CLTRS[15]分别提高了 1.39%和 0.47%.

表 4 给出了各个算法在 BR8～BR15 上的运行结果. 对强异构问题，MLHS 的优势更大，所有数据上的结果均超过比较的算法. 当只考虑 C1 时，MLHS 在 BR8～BR15 上的平均填充率达到了 93.59%，比当前最新的 VNS[14]、FDA[30]和最好的 CLTRS[15]分别提高了 2.13%、1.69%和 0.71%. 当 C1 和 C2 均考虑时，比当前较新的 HSA[29]和最好的 CLTRS[15]分别提高了 2.91%和 1.01%.

① OR-Library. http://people.brunel.ac.uk/~mastjjb/jeb/info.html

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	Mean
H_BR[17]	C1&C2	80.1	78.03	76.53	75.08	74.37	73.56	73.37	73.38	75.55
GA_GB[8]	C1&C2	87.52	86.46	85.53	84.82	84.25	83.67	82.99	82.47	84.71
TS_BG[9]	C1&C2	87.11	85.76	84.73	83.55	82.79	82.29	81.33	80.85	83.55
GRASP[12]	C1	90.26	89.50	88.73	87.87	87.18	86.70	85.81	85.48	87.69
A_2[31]	C1	88.41	88.14	87.9	87.88	87.92	87.92	87.82	87.73	87.97
maximal-space[13]	C1	91.02	90.46	89.87	89.36	89.03	88.56	88.46	88.36	89.39
VNS[14]	C1	92.78	92.19	91.92	91.46	91.2	91.11	90.64	90.38	91.46
HSA[29]	C1&C2	90.56	89.7	89.06	88.18	87.73	86.97	86.16	85.44	87.98
CLTRS[15]	C1	93.70	93.44	93.09	92.81	92.73	92.46	92.40	92.40	92.88
	C1&C2	92.26	91.48	90.86	90.11	89.51	88.98	88.26	87.57	89.88
FDA[30]	C1	92.92	92.49	92.24	91.91	91.83	91.56	91.30	91.02	91.9
MLHS	C1	94.54	94.14	93.95	93.61	93.38	93.14	93.06	92.90	93.59
	C1&C2	93.12	92.48	91.83	91.23	90.59	89.99	89.34	88.54	90.89

此外，对 BR1～BR15 数据，当考虑约束 C1 时，MLHS 算法(CPU 为 3.16 GHz)的平均运行时间为 197.33 s，而 CLTRS(CPU 为 2.6 GHz)的平均运行时间为 320 s. 当考虑约束 C1 和 C2 时，MLHS 的平均时间为 187.26 s，而 CLTRS 的平均时间为 320 s. 相对来说，MLHS 算法所需要的时间更短.

由于三维装箱问题具有很大的实用价值，许多研究者对其进行研究，填充率越来越接近最优，因此提高填充率已经越来越困难，但是 MLHS 对 BR 数据集，当考虑 C1、C1&C2 时，仍然比当前最好的算法分别平均改进 0.52%、0.74%. 这表明多层启发式搜索算法求解三维装箱问题非常有效，同时我们也观察到解的优秀程度在一定意义上是可以被继承的，即一个优秀解很可能导出另一个优秀解. 这个特点可以描述为：虽然 k 层上的最优解不一定是 $k+1$ 层上最优解的父节点；但是，k 层上最优的 N 个解很可能包括了所有 $k+1$ 层上最优的 N 个解的父节点. 这就是为什么多层启发式搜索算法比基于整数拆分的树状搜索算法更优秀的关键原因，只选择一个最优解有可能错过下一层的最优解，而选择多个最优的解进行后续搜索将以更大的概率包含下一层的最优解.

4 结束语

在文献[15,29]研究工作的基础上，本文提出了一个非常有效的多层启发式搜索算法. 该算法引入了复合块的概念，并对其加以限制，同时提出了基于多层搜索的块选择算法，使得对于装载方案的评估更加准确而迅速，从而得到很好的计算结果.

当然，由于装箱问题的复杂性以及启发式算法本身的缺陷，本文算法还有一些不足之处. 例如，随着实例规模的增大，复合块的生成方式更多，搜索树也会更复杂，可能需要较长的计算时间. 因此，将来的工作是进一步优化算法，提高计算速度，同时，结合实际的应用，考虑更多的约束条件.

参 考 文 献

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

[2] Scheithauer G. Algorithms for the container loading problem//Operations Research Proceedings. Berlin: Springer, 1992: 445-452

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

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

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

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

[7] 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

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

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

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

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

[12] Moura A, Oliveira J F. A GRASP approach to the container-

loading problem. IEEE Intelligent Systems, 2005, 20(4): 50-57

[13] Parreño 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

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

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

[16] 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

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

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

[19] Sixt M. Dreidimensionale Packprobleme. Lösungsverfahren basierend auf den Meta-Heuristiken Simulated Annealing und Tabu-Suche. Frankfurt am Main: Europäischer Verlag der Wissenschaften, 1996

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

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

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

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

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

[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] 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)

[27] 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)

[28] 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)

[29] 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)

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

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

ZHANG De-Fu, born in 1971, Ph. D., professor. His research interests include computational intelligence and data mining.

PENG Yu, born in 1984, Ph. D. candidate. His research interests include algorithm and computational intelligence.

ZHANG Li-Li, Ph. D. candidate. Her research interest is multi-objective optimization.

Background

The work is supported by the National Nature Science Foundation of China (Grant No. 61272003). The projects aim at designing highly efficient algorithms for NP hard problems, such as the packing problem and timetabling problem, which are the most important part of automated packing and timetabling software. The project group has designed some efficient algorithms for the rectangle packing problem, three-dimensional packing problem, two-dimensional irregular packing, routing problem and timetabling problem, some results have been published. The fundamental aim of this paper was to investigate three-dimensional packing problem and to develop state-of-the-art algorithms that could be used in an industrial field. The paper belongs to one of the key parts of the project.