文章编号: 1001-0920 (2014) 02-0373-06 **DOI:** 10.13195/j.kzyjc.2012.1552

集装箱堆场预倒箱问题的混合优化算法

边 展[1], 李 娜[1], 李向军[2], 靳志宏[1]

(1. 大连海事大学 交通运输管理学院，辽宁 大连 116026；2. 大连海洋大学 信息工程学院，辽宁 大连 116023)

摘 要: 堆场出口箱区通常通过集装箱的预倒箱操作来提高后续装船作业的效率. 为此, 开发了由邻域搜索算法与整数规划算法组成的两阶段混合算法对预倒箱问题进行优化, 第 1 阶段通过启发式规则压缩末终堆存状态空间, 第 2 阶段通过整数规划算法缩短第 1 阶段得到的预倒箱序列的长度. 两个阶段循环交替进行以快速求得最优的预倒箱序列. 借助不同种类仿真算例的实验结果及与现有研究方法下所得结果的对比, 验证了混合优化算法的有效性和实用性.

关键词: 集装箱；预倒箱操作；压箱数；邻域搜索算法；整数规划

中图分类号: U169.62 **文献标志码:** A

Hybrid optimization algorithm for pre-marshalling export containers

BIAN Zhan[1], *LI Na*[1], *LI Xiang-jun*[2], *JIN Zhi-hong*[1]

(1. College of Transportation Management，Dalian Maritime University，Dalian 116026，China；2. College of Information Engineering，Dalian Ocean University，Dalian 116023，China. Correspondent：BIAN Zhan，E-mail: bianzhan1990@163.com)

Abstract: Extra re-handles may occur when lifting containers up for loading onto ships. One way to improve loading efficiency is to pre-marshal the containers in such an order that it fits the loading sequence. Therefore, a two-stage algorithm composed of a neighborhood search algorithm and an integer programming model is proposed to develop a pre-marshalling plan to improve the layout of containers in a bay. In the first stage, the final layout state space is compressed by using heuristic rules; in the second stage, the length of pre-marshalling sequence is shortened by integer programming. Two stages execute alternately to get optimal pre-marshalling sequence quickly. Several sets of experimental results demonstrate the effectiveness and practicability of the hybrid optimization algorithm.

Key words: container；pre-marshalling containers；misoverlays；neighborhood search algorithm；integer programming

0 引 言

出口箱区的取箱操作是集装箱装船作业中的重要环节. 为了有效利用堆场空间资源, 集装箱通常采取多层堆码的方式进行存储. 由于执行取箱作业时, 龙门吊只能从最上层开始提取集装箱, 若堆存层级状态与取箱顺序不一致, 则会造成压箱现象, 导致取箱时需要额外的翻箱操作, 从而使作业成本上升. 因此, 为提高装船效率, 码头在装船前通常需进行预倒箱作业.

Kang 等[1]结合偏序图与分支定界法, 运用模拟退火算法求得预倒箱操作计划表, 但由于文中集装箱是移至新贝而并非在原贝内进行操作, 实用性不佳. Lee 等[2]以最小化预倒过程中的倒箱量为目标, 提出了相应的整数规划模型和启发式方法, 对倒箱路径进行了优化. Caserta 等[3]提出采用 Corridor 算法对预倒箱问题进行研究. Huang 等[4]将预倒箱问题分为类型 A 和类型 B 两种, 并针对两种问题分别开发了启发式算法进行求解. Bortfeldt 等[5]开发一种启发式树搜索算法, 并将求解结果与现有的研究结果进行了对比分析. 随后, Forste 等[6]改进了树搜索算法, 设计了更精细的移动分类策略与不同的分支定界规则, 但仍需借助贪婪启发式算法进行求解. Rodriguez-Molins 等[7]运用域依赖的启发式策略解决预倒箱问题, 但随着问题规模的增大, 约束条件相应增多, 导致

收稿日期: 2012-10-17；**修回日期:** 2013-01-26.

基金项目: 国家自然科学基金项目(71172108)；大连市科技计划项目(2012A17GX125)；中央高校基本科研业务费专项资金项目(3132013320)；高等院校博士点基金项目(20122125110009, 20132125120009).

作者简介: 边展(1990−), 女, 博士生, 从事物流系统优化的研究；靳志宏(1963−), 男, 教授, 博士生导师, 从事物流系统优化等研究.

求解效率及质量降低. 国内针对此领域的研究相对较少, 董琳等[8]利用图论知识构建数学模型, 并用加以限制的广度搜索算法计算出翻箱的最少步骤, 但是文中未给出具体实例说明及优化结果; 白治江等[9]建立了用网络结构描述的整数规划模型, 使用分支定界法与启发式相结合的两阶段法进行求解; 易正俊等[10]将翻箱优化问题转化为最短路径求解问题, 运用PCNN优化控制算法获得最优的翻箱方案.

由文献[2]可知, 运用网络模型可求得预倒箱问题的最优解, 但求解时间也将随着问题规模的扩大而增加, 这样可能会延长集装箱船在港停留时间而徒增不必要的费用. 为此, 本文构建了包含整数规划模型在内的基于邻域搜索算法的启发式算法对预倒箱问题进行处理, 力求在较短的时间内求得大规模问题的最优解.

1 问题描述与假设条件

集装箱码头堆场由多个街区组成, 某街区的结构如图1所示. 沿Z轴方向堆存集装箱的垂直区域称为列/栈; 沿Y轴方向若干栈所形成的区域称为贝; 沿X轴方向若干栈排列所形成的区域称为行; 街区沿XY构成的二维平面展开, 所构成的平面称为层, 每个集装箱对应的位置称为箱位. 目前, 堆场中普遍使用龙门吊进行取箱作业, 每台龙门吊上配有一台小车, 小车上装有夹箱器.

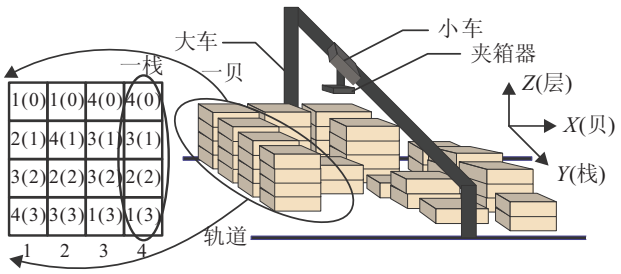


图 1 码头堆场某街区示意图

由于出口箱有“随机到达、整批离开”的特性, 可能导致其堆存顺序与装船顺序不一致, 通常会有需要较早装船的集装箱堆放在较晚装船的集装箱的下方, 造成压箱现象. 集装箱预倒箱作业是指出口集装箱在装船作业开始之前, 依据预倒箱序列通过集装箱的搬移操作, 使堆场由初始堆存状态转化为末终堆存状态. 搬移集装箱需消耗时间和成本, 而末终堆存状态将影响后续集装箱装船作业的效率; 前者可由预倒箱序列的长度来衡量, 后者可用压箱数作为评价标准.

预倒箱问题可归纳为: 在一定范围内的堆场中, 在贝位数量、额定高度及初始堆存状态已知的情况下, 求解如何以最少的集装箱搬移次数, 达到压箱数较少的末终堆存状态.

基于现场作业实际, 为求解方便做出如下约定:

1) 预倒箱操作发生在同一贝中. 作业进行时, 龙门吊大车静止, 仅小车移动.

2) 仅考虑一台龙门吊的作业.

3) 集装箱的装船顺序已知.

4) 在不同栈之间搬移集装箱的时间相等.

5) 不同尺寸的集装箱堆放在不同的贝.

6) 仅考虑即将抵达的集装箱船, 不考虑随后的船次, 即不属于该船次的集装箱彼此间不考虑可能产生的压箱现象.

7) 某贝在进行预倒箱作业过程中, 没有其他集装箱进入.

2 预倒箱作业描述

2.1 预倒箱序列

将集装箱由某一栈夹起或在某一栈放下的过程称为搬移操作. 一个搬移操作可由(起点栈,讫点栈)或(起点栈, 讫点栈, 箱号)两种形式描述, 前者称为基本搬移操作, 后者称为完整搬移操作. 由多个集装箱搬移操作所构成的有序集合称为预倒箱序列, 根据其构成内容的不同可分为以下两种:

1) 基本预倒箱序列. 由多个基本搬移操作所构成的有序集合称为基本预倒箱序列. 一个基本预倒箱序列若包含以下两种情况中的任意一种, 即为不可行序列:

① 执行某搬移操作时, 该操作的起点栈内并无集装箱;

② 执行某搬移操作时, 该操作的讫点栈已达额定高度.

2) 完整预倒箱序列. 由多个完整搬移操作所构成的有序集合称为完整预倒箱序列. 一个完整预倒箱序列若包含以下3种情况中的任意一种, 即为不可行序列:

① 执行某搬移操作时, 该操作欲移动的集装箱并未存放在起点栈中;

② 执行某搬移操作时, 该操作的讫点栈已达额定高度;

③ 执行某搬移操作时, 该操作欲移动的集装箱被压在一个或多个集装箱下方.

预倒箱序列的长度由序列中搬移操作的数量来衡量. 若一个基本预倒箱序列表示为$\{(m,n),(i,j),(p,q)\}$, 则该序列长度为3.

显然, 预倒箱序列具备以下两个性质:

性质 1 给定集装箱堆场的初始堆存状态及可行的预倒箱序列q, 集装箱c是堆场中的集装箱. 若将c移出堆场, 并令q'为由q中移除c所有搬移操作后所得的预倒箱序列, 则q'仍为可行的预倒箱序列.

性质 2　假设暂时松弛额定高度的限制. 给定集装箱堆场的初始堆存状态及可行的预倒箱序列 q, 集装箱 c 是堆场中的集装箱. 对应集装箱 c 产生一个虚拟集装箱 c', 并对应 c' 产生一条搬移路径. 令 q' 为将集装箱 c' 的路径加入原可行的预倒箱序列 q 中后所得的新的预倒箱序列, 若满足: 1) q' 可行; 2) c' 在起始状态中的位置位于 c 的紧上方; 3) c' 路径的最后一个预倒箱操作是 c 路径最后一个预倒箱操作的紧后动作; 4) 末终堆存状态 c' 与 c 所在栈相同, 则末终堆存状态 c' 仍位于 c 的紧上方.

2.2　压 箱 数

为分析集装箱堆场任意时刻的状态, 引入“压箱数”的概念. 压箱数是指某栈中最严重的压箱现象的深度, 即某栈压箱数 = 该栈目前堆存高度 − 发生压箱的最大深度.

设集装箱的箱号编号由 0 开始, 最大编号为 (集装箱总数 −1); 根据集装箱装船的先后顺序, 可将其划分为若干优先级, 最早装船的集装箱优先级编号设为 1, 则箱号为 m、优先级为 n 的集装箱记为 $n(m)$.

若某栈优先级编号较小的集装箱均堆放在编号较大的集装箱上面, 则无压箱现象, 即压箱数为 0. 若某贝由 1、2、3、4 栈构成, 如图 1 所示. 显然, 栈 1 的压箱数为 0. 在栈 2 中, 压箱现象发生在 4(1)、2(2) 两个箱子之间, 故压箱数 = 4(栈的堆存高度) − 2(2(2) 的高度) = 2. 栈 3 的压箱现象发生在 4(0) 和 3(1) 及 3(2) 和 1(3) 两组箱子之间, 故压箱数 = max{[4(栈的堆存高度) − 3(3(1)的高度)], [4(栈的堆存高度) − 1(1(3)的高度)]} = 3. 在栈 4 中, 4(0) 与 3(1)、3(1) 与 2(2)、2(2) 与 1(3) 均发生压箱现象, 由于此现象最大深度为 1(1(3)的高度), 故压箱数为 3. 该贝的压箱数为 $8(0+2+3+3)$.

针对集装箱搬移操作的数量及末终堆存状态的压箱数两项指标, 将一个预倒箱序列的长度与其所对应的末终堆存状态压箱数加权相加设定为目标函数, 作为该预倒箱序列优劣评估的依据, 两者之比设为 $W_1 : W_2 = 0.1 : 1$.

3　两阶段混合算法设计

混合算法包含以下两个阶段: 第 1 阶段, 运用基于阈值接受启发式规则的邻域搜索算法求得末终堆存状态压箱数较少的预倒箱序列; 第 2 阶段使用二元整数规划模型, 在保证末终堆存状态不变的前提下缩短第 1 阶段得到的预倒箱序列的长度. 两个阶段循环交替进行, 以便快速求得最优的预倒箱序列.

3.1　第 1 阶段——邻域搜索算法

此阶段基于阈值接受启发式规则设计邻域搜索算法, 其目标是最小化末终堆存状态的压箱数. 显然, 阈值越高, 算法搜索的范围越大, 运行的时间越久; 阈值越低, 虽然缩短了运行时间, 但求解结果有早熟的风险. 参考以往计算经验, 取初始目标函数值的 1/4 作为初始阈值.

尽管缩短预倒箱序列长度不是该阶段的重点, 但有必要避免序列过长的情况, 故目标函数用 2.2 节中提到的加权值表示. 另外, 该阶段采用基本预倒箱序列, 定义 A 为阈值, T 为当前预倒箱序列, T^* 为最优可行预倒箱序列, F 为 T 所对应的目标函数值, F^* 为 T^* 所对应的目标函数值, 算法步骤表述如下.

Step 1: 初始化. 设初始预倒箱序列为 T^*, 设定初始阈值为 F^* 的 1/4, $T = T^*$.

Step 2: 产生新解. 通过以下 3 种方法随机变动当前解以产生新解: ① 以 P_1 的概率随机插入一搬移操作; ② 以 P_2 的概率删除一随机选择的搬移操作; ③ 以 P_3 的概率随机选择两个搬移操作并对调两者的位置.

Step 3: 可行性检验. 若 T 不可行, 则转至 Step 2, 通过删除操作等方法将其变为可行.

Step 4: 阈值接受检验. 若 $F < F^* + A$, 则接受 T 为最优可行预倒箱序列, 即 $T^* \leftarrow T$; 当每进行 3 000 次循环或连续 100 次循环未能求得更优的可行解时, A 降低 5%.

Step 5: 算法终止. 若 $A < 1$, 算法终止, 输出 T^*, 否则转至 Step 2.

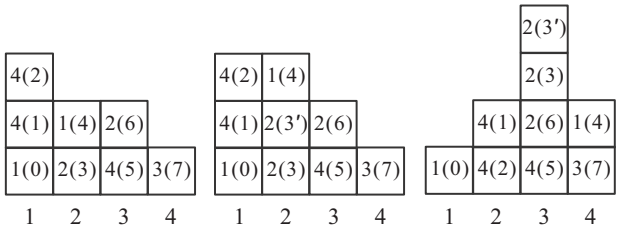
值得注意的是, 因为预倒箱序列通常会在 Step 3 中因删除操作而变短, 所以概率 P_1 应远大于概率 P_2、P_3 的取值. P_1 值越低, 算法运行速度越快, 但会影响最终解的质量, 故设 P_1、P_2、P_3 的值分别为 0.89、0.10、0.01.

3.2　第 2 阶段——整数规划

此阶段采用完整预倒箱序列, 故在阶段开始时, 需通过模拟的方法, 记录每个搬移操作所搬动的集装箱的编号, 将第 1 阶段所得的基本集装箱预倒箱序列转换为含箱号集装箱预倒箱序列. 之后, 运用 2.1 节中预倒箱序列的两个性质, 构建二元整数规划模型以缩短预倒箱序列的长度.

图 2 举例说明了虚拟集装箱的生成及其搬移路径的设定规则. 图 2(a) 为贝中集装箱的初始堆存状态, 由此可得初始可行的预倒箱序列为 $q = \{(1,3,2),(2,4,4),(2,1,3),(3,2,2),(1,3,3),(1,2,1)\}$. 根据性质 2, 对 2 栈中的集装箱 2(3) 增加一个虚拟箱 2(3′), 从而初始堆存状态转化为图 2(b). 对集装箱 2(3′) 随机产生搬移路径, 并加入到预倒箱序列 q 中, 生成新

的预倒箱序列 q'. 其中, 虚拟集装箱 2(3′) 的最后一步搬移动作置于集装箱 2(3) 的最后搬移动作之后, $q = \{(1,3,2), (2,4,4), (2,4,3'), (2,1,3), (3,2,2), (1,3,3), (1,3,3'), (1,2,1)\}$, 末终堆存状态如图 2(c) 所示.



(a) 初始状态 (b) 处理后的初始状态 (c) 末终状态

图 2 虚拟箱及搬移路径示意图

定义堆存区初始状态为 Y, 其区域内堆存集装箱的集合为 C. 令 q 为一个可行的预倒箱序列, Y_q 为 Y 对应于 q 的末终状态. 根据性质 2, 在松弛额定高度限制的前提下对堆存区的每一个集装箱 c, 产生一定数目的虚拟集装箱 $c', c'', \cdots, c^{(n)}$, 并对应设定各虚拟集装箱的搬移路径. 同时令初始状态中集装箱 c' 位于集装箱 c 的紧接上方, c'' 位于 c' 的紧接上方, 以此类推. 令 Y' 为加入所有虚拟集装箱后集装箱堆存区的初始状态, q' 为加入所有虚拟集装箱的搬移路径所得的预倒箱序列. 需要指出的是, 此处 q' 不一定为可行预倒箱序列.

定义符号如下: x_i 为决策变量, 对于任一真实或虚拟集装箱 i, 若此集装箱被选择, 则其值为 1, 否则为 0. c_i 为真实或虚拟集装箱 i 的搬移路径长度, 即为集装箱 i 在预倒箱过程中被移动的次数. C 为所有真实集装箱的集合. C' 为所有真实与虚拟集装箱的集合. D_i 为真实集装箱 i 以及所对应的虚拟集装箱的集合. V 为预倒箱过程中某栈在某时刻所堆存的真实与虚拟集装箱的集合.

目标函数为

$$\mathrm{Min}\sum_i c_i x_i,\ i \in C'. \tag{1}$$

约束条件为

$$\sum_{i\in D_i} x_i = 1,\ i \in C; \tag{2}$$

$$x_i + x_j \leqslant 1,\ \text{paths } i \text{ and } j \text{ conflict}; \tag{3}$$

$$\sum_{i\in V} x_i \leqslant H,\ V \text{ an over} - \text{height conflict set}; \tag{4}$$

$$x_i \in \{0,1\},\ i \in C'. \tag{5}$$

目标函数 (1) 表示最小化集装箱搬移次数, 约束 (2) 保证每一个真实集装箱的可能替代路径只选择一个, 约束 (3) 为排除搬移冲突, 约束 (4) 确保集装箱堆存的高度不超过栈的额定高度, 约束 (5) 为决策变量 0-1 约束.

此模型的一组可行解为由每一个真实集装箱 i 所对应集合 D_i 中各选一条搬移路径组成, 且在整个预倒箱过程中不会产生冲突或堆放超高的预倒箱序列. 此模型的最优解为所有可行解中搬移次数最少的解. 由性质 1 和性质 2 可知, 模型的可行解所对应的末终堆存状态均为 Y_q. 此外, 此模型所依据的可行预倒箱序列 q 本身即为此模型的可行解, 故此模型的可行解必然存在.

3.3 两阶段混合算法流程

运用两阶段启发式算法进行求解时, 整体循环上限定为 N_1, 3.1 节所述第 1 阶段的 Step 2 中新解产生无循环上限, 仅以阈值作为约束标准; 而第 2 阶段循环次数设定为 N_2 次. 算法流程如图 3 所示.

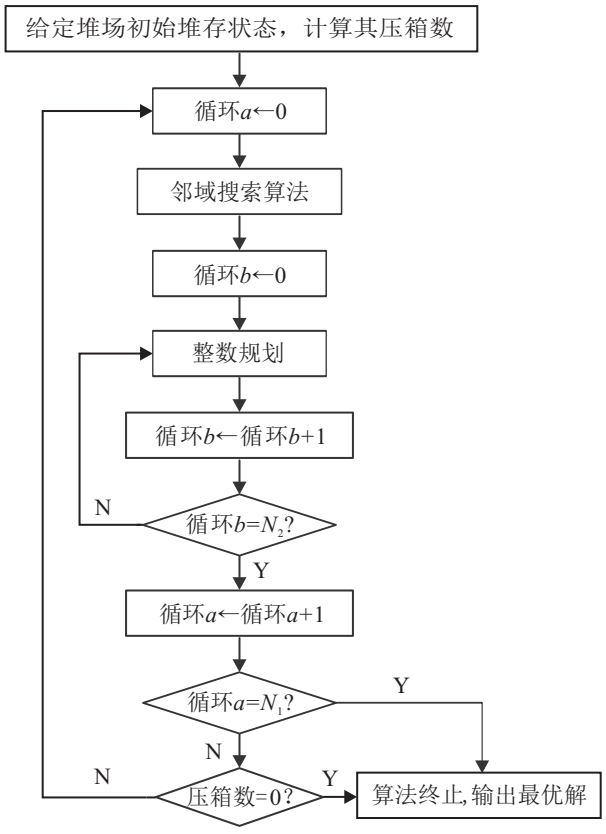


图 3 两阶段混合算法示意

4 实验及结果分析

实验运算环境为中央处理器 (CPU)Intel Pentium T4 200, RAM 为 512 MB, Windows XP 的 PC. 设定 $N_1 = 30, N_2 = 3$, 第 2 阶段的二元整数规划, 运用了 Lingo 进行求解. 算法结束后, 可获得目标函数值最小的预倒箱序列、对应的压箱数、搬移操作序列长度和经过这些搬移操作之后, 堆存区域的末终状态.

参数表示如下: S 为堆场中栈的数量, H 为额定堆存高度, N 为堆场中集装箱的总数量, K 为堆场中集装箱的种类, 基于给定的 S、H、N、K, 随机产生堆场的初始堆存状态.

为探讨不同的参数对模型的影响, 设计了多种不同性质的实验, 求解结果及分析如下.

4.1 不同性质的实验及结果分析

首先, 随机生成不同规模的实验数值例 (堆存区域利用率均为 70%), 按照两阶段启发式算法进行计算, 所得结果如表 1 所示. 可以看出, 运算结果中的各种指标数值随着堆存区域的增大而增大. 当堆存区域容量相同时 (4×10 与 5×8), 尽管额定堆存高度不同, 结果却非常接近; 而当栈数或额定堆存高度其中一个增加时, 求解结果也正比增加. 此外, 综合表中所列出的第 1 阶段求得初始预倒箱序列的循环次数与算法求解时间可知, 两阶段混合算法的收敛效果良好.

其次, 将利用率设定为不同数值, 以测试对压箱数的影响. 堆存区域大小设定为 5×13, 所得结果如表 2 所示. 可以看出, 在堆存区域容量一定的情况下, 初始压箱数随着堆存区域利用率的提高而增加, 在利用率提高到 79% 后, 便无法求得末终压箱数为 0 的解. 预倒箱序列求解的时间在利用率提高到 79% 后呈现与利用率相反的变化趋势, 这是因为当利用率为 79% 时, 堆存区域容量为 65 个集装箱, 而实际堆存有 51 个集装箱, 可供预倒箱的位置仅剩 14 个, 翻倒作业将不易进行. 而邻域搜索算法结束后, 二元整数规划会将所求得的预倒箱序列缩短, 导致算法很快陷入局部搜索, 此局部最优解会保持到最后成为最终结果. 但通过数值例 5～例7 (堆存区域利用率分别为 85%、89%、94%) 可以看出, 算法降低高利用率堆场的压箱数的效果还是非常明显的 (压箱分别降低了 20%、23%、33%).

最后, 将堆存区域的大小设定为 5×13, 堆存区域利用率定为 70%, 即共堆存 45 个集装箱. 现假定集装箱的总数不变, 改变集装箱优先级的个数, 取 10 种不同的集装箱优先级进行测试, 所得结果如表 3 所示. 可以看出, 末终压箱数、预倒箱序列长度和算法运行时间均没有随着集装箱优先级的细分而发生明显变化, 可以推断, 集装箱优先级的细分程度对预倒箱影响不大. 因此, 对于集装箱装船作业而言, 只要依据装船顺序表, 令每一个集装箱为一个优先级, 则可以求得满意的预倒箱序列.

表 1　不同规模的实验运算结果

规模 $H \times S$	箱量 N	初始压箱数	末终压箱数	压箱数降低/%	第 1 阶段循环次数	预倒箱序列长度	算法耗时/s
4×8	23	13	0	100	33 909	18	48.516
4×10	28	18	0	100	47 000	28	70.313
5×8	28	18	0	100	54 362	27	69.703
5×10	35	23	0	100	77 000	33	85.421
5×13	45	25	0	100	61 796	32	78.875
5×16	56	31	0	100	81 000	38	102.437
9×13	82	60	28	53	115 000	49	124.437
9×16	100	72	39	46	128 000	50	136.000

表 2　不同利用率的实验运算结果

堆存利用率/%	箱量 N	初始压箱数	末终压箱数	压箱数降低/%	预倒箱序列长度	算耗时/s法
49	32	14	0	100	18	74.016
60	39	19	0	100	23	73.828
70	45	25	0	100	32	78.875
79	51	31	4	87	35	81.562
85	55	35	20	43	19	65.938
89	58	37	23	38	19	66.250
94	61	40	33	18	9	64.671

表 3　不同优先级的实验运算结果

优先级数量	初始压箱数	末终压箱数	压箱数降低/%	预倒箱序列长度	算法耗时/s
3	23	0	100	28	72.000
4	26	0	100	33	80.626
5	20	0	100	27	73.719
7	23	0	100	28	76.296
8	26	0	100	36	84.781
10	25	0	100	32	78.875
20	29	0	100	36	80.343
25	30	0	100	45	102.297
30	31	2	94	37	87.688
45	33	0	100	46	88.875

4.2 与现有研究的比较分析

针对文献 [2] 中降低压箱数模型, 通过同种堆场堆存状态的算例进行测试对比.

取文献 [2] 扩展模型中的第 1 个数值例 ($S = 10$, $H = 4, N = 25, K = 3$) 进行测试. 根据压箱数的定义, 此例初始压箱数为 10. 通过文献 [2] 求得的末终压箱数为 0, 预倒箱序列长度为 12, 求解时间为 13 444 s. 通过两阶段启发式算法求得的末终压箱数为 0, 预倒箱序列长度为 16, 求解时间为 60.563 s.

将文献 [2] 扩展模型中的其他两个数值例以及新生成的数值例运用文献 [2] 的方法及两阶段启发式算法分别求解, 与上述实验结果汇总如表 4 所示.

表 4 不同数值例的实验运算结果

数值例	初始	末终压箱数		序列长度		算法耗时/s	
(H, S, N, K)	压箱数	文献 [2]	本文	文献 [2]	本文	文献 [2]	本文
(4,10,25,3)	10	0	0	12	16	1 344.000	60.563
(4,16,35,4)	13	0	0	15	19	195.000	89.110
(4,15,31,31)	10	0	0	12	17	17 210.000	80.656
(4,12,29,20)	12	0	0	13	18	11 120.000	75.526
(4,14,30,15)	13	0	0	15	19	8 510.000	80.635
(4,14,25,10)	12	0	0	12	17	4 822.000	71.716

由表 4 可以看出, 两阶段启发式算法求得的预倒箱序列的长度稍长于文献 [2], 而求解时间却明显短于文献 [2], 可见此算法可有效提高求解效率.

集装箱堆场实际作业中, 作业时间长短是至关重要的因素, 这要求优化算法以最短的时间求得最优的结果. 两阶段启发式算法求解数值例的时间均在 3 min 以内, 鉴于求解快速且求得结果优良, 故较文献 [2] 的算法更具可行性和现实意义.

5 结 论

针对集装箱堆场出口箱预倒箱问题, 以降低压箱数为目的, 开发了基于阈值接受法的两阶段启发式算法, 从而求得优质的预倒箱序列以指导倒箱作业的进行. 所采取的实验均以实务为参照标准. 实验结果表明, 运用两阶段启发式算法均可在合理的时间内完成运算过程, 并可获得优良的预倒箱序列, 具有良好的可行性.

文中预倒箱范围设定在同一贝内, 未来可同时考虑龙门吊大车的移动, 将预倒箱作业拓展至整个堆场.

参考文献(References)

[1] Jaeho Kang, Myung-Seob Oh, Eun Yeong Ahn, et al. Planning for intra-block remarshalling in a container terminal[C]. IEA/AIE 2006. Berlin Herdelberg, 2006: 1211-1220.

[2] Lee Y S, Hsu N Y. An optimization model for the container pre-marshalling problem[J]. Computers and Operations Research, 2007, 34(11): 3295-3313.

[3] Caserta M, Schwarze S, Voβ S. A corridor method-based algorithm for the pre-marshalling problem[C]. Proc of the EvoWorkshops. Tübingen, 2009: 788-797.

[4] Huang S H, Lin T H. Heuristic algorithms for container pre-marshalling problems[J]. Computers and Industrial Engineering, 2012, 62(1): 13-20.

[5] Bortfeldt A, Forster F. A tree search procedure for the container pre-marshalling problem[J]. European J of Operational Research, 2012, 217(3): 531-540.

[6] Forster F, Bortfeldt A. A tree search procedure for the container relocation problem[J]. Computers and Operations Research, 2012, 39(2): 299-309.

[7] Rodriguez-Molins M, Salido M A, Barber F. Intelligent planning for allocating containers in maritime terminals[J]. Expert Systems with Applications, 2012, 39(1): 978-989.

[8] 董琳, 刘庆敏, 王超, 等. 集装箱翻箱问题的模型分析及算法[J]. 经济数学, 2006, 23(2): 181-186.
(Dong L, Liu Q M, Wang C, et al. Modeling and algorithm for turning out problem of container[J]. Mathematics in Economics, 2006, 23(2): 181-186.)

[9] 白治江, 王晓峰. 集装箱翻箱优化方案设计[J]. 水运工程, 2008, 4(414): 57-61.
(Bai Z J, Wang X F. An optimal solution design for container re-handling[J]. Port and Waterway Engineering, 2008, 4(414): 57-61.)

[10] 易正俊, 江静, 胡勇. 堆场集装箱翻箱的PCNN优化控制算法[J]. 自动化学报, 2011, 37(2): 241-244.
(Yi Z J, Jiang J, Hu Y. An optimization control algorithm for containers relocation based on PCNN model[J]. Acta Automatic Sinica, 2011, 37(2): 241-244.)

(责任编辑: 孙艺红)