文章编号: 1001-0920 (2013) 11-1707-06

热轧板坯出库问题的树搜索算法

张瑞友, 刘士新, 汪定伟

(东北大学 a. 信息科学与工程学院，b. 流程工业综合自动化国家重点实验室，沈阳 110819)

摘 要: 热轧板坯的出库问题是连铸-热轧生产中一个重要的组合优化问题, 然而在学术界还很少见到对该问题的研究. 对此, 提出了热轧板坯出库问题总移动次数的一个下界, 开发了一个极小化总移动次数的树搜索算法. 该算法包括一个生成初始解的贪婪算法和一个基于复合移动的递归搜索. 大量的实验和分析表明, 该树搜索算法能在较短的时间内给出板坯出库问题的满意解, 具有重要的理论意义和应用价值.

关键词: 板坯出库问题；连铸-热轧；树搜索；组合优化；倒箱问题

中图分类号: TP29 **文献标志码:** A

Slab retrieving problem in hot rolling and its tree search algorithm

ZHANG Rui-you, *LIU Shi-xin*, *WANG Ding-wei*

(a. College of Information Science and Engineering，b. State Key Laboratory of Synthetical Automation for Process Industry，Northeastern University，Shenyang 110819，China．Correspondent：ZHANG Rui-you，E-mail: zhangruiyou@ise.neu.edu.cn)

Abstract: Slab retrieving problem is an important combinatorial optimization problem in continuous casting and hot rolling. However, it has been seldom researched in literature. Therefore, a lower bound of total number of moves for the slab retrieving problem is proposed. A tree search algorithm minimizing the total number of moves is developed. The tree search algorithm includes a greedy search for an initial feasible solution and a recursive search based on compound moves. Numerous experiments and analyses show that the tree search algorithm can provide satisfactory solutions of the slab retrieving problem in relatively short running time and therefore has important theoretical meaning and practical value.

Key words: slab retrieving problem；continuous casting and hot rolling；tree search；combinatorial optimization; container relocation problem

0 引 言

炼钢-连铸-热轧生产过程中的计划与调度问题多年来在学术界一直备受关注. 很多学者针对该生产过程中的合同计划问题[1]、余材匹配问题[1]、作业计划问题[2], 以及多种调度问题[3-4]展开了研究, 并取得了大量的学术成果. 板坯库是连接连铸和热轧环节的中间产品库, 其上游的连铸环节按浇次组织生产, 其下游的热轧环节按轧制批量组织生产. 连铸和热轧组织方式的不同导致了板坯入库和出库方式的不同, 板坯在复杂的约束条件下以不同的方式入库和出库, 这为板坯库的高效管理提出了严峻的挑战.

热轧板坯库一般包括若干库区和一个步进式上料机. 每个库区包括若干行板坯垛位和2 ～ 3台吊机, 每个垛位最多可堆放十几块板坯, 吊机用于将板坯在不同的垛位之间移动或将板坯运至上料机. 根据吊机型号的不同, 一次可移动一块或多块板坯. 板坯运至上料机后等待进入加热炉, 称为板坯的出库. 轧制计划决定了板坯的出库顺序, 然而板坯的堆垛顺序又决定了只有位于垛顶的板坯才可以出库. 尽管在入库时会尽量考虑板坯的出库顺序, 但由于连铸和热轧组织方式不同等因素, 板坯的堆垛顺序往往与出库顺序不同. 如果待出库的板坯上面堆放有其他板坯, 则在出库前必须先将其他板坯移到其他垛位, 这称为倒垛. 如何以最小的成本完成板坯的出库(包括必要的倒垛)称为板坯的出库优化问题.

与本文研究的板坯出库问题相关的是编制轧制

收稿日期: 2012-11-23；**修回日期:** 2013-03-29.

基金项目: 国家自然科学基金项目(71001019, 71171038, 70931001, 71021061)；中国博士后科学基金项目(2012M520642)；中央高校基础科研业务费项目(N120404016).

作者简介: 张瑞友(1979−), 男, 副教授, 博士, 从事建模与优化、物流系统等研究；刘士新(1968−), 男, 教授, 博士生导师, 从事生产计划与调度、物流建模与优化、智能优化算法等研究.

计划时的板坯选择问题. 文献 [5-6] 提出了板坯选择问题的数学模型以及启发式求解算法; 文献 [7-8] 将库区的吊机能力引入板坯选择问题, 使板坯选择问题的解更接近实际; 文献 [9] 也研究了类似的板坯选择问题并设计了一个并行遗传算法. 上述研究与本文在生产组织方面相关, 但本质上属于不同的问题.

与本文问题相似的是集装箱码头操作中的倒箱和预倒箱问题. 文献 [10] 针对集装箱倒箱问题提出了一种启发式算法; 文献 [11-12] 针对倒箱问题提出了一种走廊算法; 文献 [13] 为倒箱问题开发了一种树搜索算法, 并取得了很好的效果; 关于倒箱和预倒箱问题的类似研究还有文献 [14-17] 等. 但是, 集装箱倒箱 (以及预倒箱) 问题中堆垛的最大高度比板坯垛矮, 一般只有 $6 \sim 8$ 层, 而且目前公开发表的文献中都假设一次只能移动一个集装箱.

本文以一次可移动多块板坯的板坯出库问题为研究对象, 给出了问题的数学描述, 提出了总移动次数的一个下界, 并借鉴文献 [13] 的基本思想, 开发了一种高效的树搜索算法. 通过大量的实验, 对该树搜索算法进行了深入的验证和分析. 结果表明, 本文提出的算法是合理有效的, 能够快速给出一个较优的出库方案, 从而降低倒垛成本, 提高经济效益, 具有理论和应用价值.

1 板坯出库问题的数学描述

本文研究的板坯出库问题可描述如下: 一个轧制批量计划涉及一定数量的板坯垛位, 每个板坯垛位有最大堆垛高度, 每块板坯有唯一编号. 板坯可在堆垛之间移动, 或由堆垛向步进式上料机移动, 每次移动的最大板坯数量给定. 由堆垛向上料机移动时, 板坯的编号应是堆垛中所有板坯编号中最小的, 如果向上料机一次移动多块板坯, 则移动的每一块板坯都应是除了其上面的板坯之外堆垛剩余板坯中编号最小的. 假设在任意堆垛之间或由堆垛向上料机移动一次板坯的成本相同, 问题的目标是极小化总移动次数.

数学上该问题可描述如下: 给定 S 个栈, 每个栈的最大容量为 H, 各栈中有若干元素, 每个元素都有一个唯一的编号. 初始时栈 $s(s = 0, 1, \cdots, S-1)$ 中有 $h_s (h_s \leqslant H)$ 个元素, 栈 s 的第 $i (i = 0, 1, \cdots, h_s - 1$ 自栈底向栈顶递增) 个元素的编号为 $g_{si} (g_{si} > 0)$. 元素可在任意两个栈之间“移位”, 当前所有栈中编号最小的元素可从栈中“移除”, 移位和移除统称“移动”, 单次移动中可以移动的最大元素个数为 M. 注意到如果一次移除多个元素, 除了移除的最顶上元素是当前所有栈中编号最小的元素以外, 下面的任一元素也是除了其上面元素外所有栈中编号最小的. 问题的目标是最小化可以清空所有栈的总的移动次数.

板坯出库问题中任一时刻的布局可以用如下矩阵描述:

$$\boldsymbol{L} = \begin{bmatrix} g_{0,H-1} & \cdots & g_{S-1,H-1} \\ \vdots & \ddots & \vdots \\ g_{0,0} & \cdots & g_{S-1,0} \end{bmatrix}.$$

其中: 非零元素表示相应的编号, “0”表示相应位置无元素. 任一移动可以用如下三元组描述:

$$m = f \xrightarrow{n} t. \tag{1}$$

其中: f 和 t 分别表示移动的源栈和目的栈, n 表示移动的元素个数. 如果 $t = -1$, 则 m 为移除, 否则 m 为移位. 例如, 某板坯出库问题可用如下矩阵描述:

$$\boldsymbol{L} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 8 & 0 \\ 6 & 10 & 4 & 11 \\ 7 & 12 & 9 & 5 \end{bmatrix}. \tag{2}$$

其中: $S = 4, H = 5$. 设 $M = 3$, 则当前布局下可行的移位包括 $2 \xrightarrow{1} 3, 2 \xrightarrow{3} 3$ 等; 可行的移除为 $0 \xrightarrow{1} -1$, 实施该移除后出现新的可行移除 $1 \xrightarrow{2} -1$.

2 板坯出库问题的下界

给定板坯出库问题的一个布局 $\boldsymbol{L}$, 如果某栈 s 中上下相邻的两个元素满足 $g_{si} < g_{s(i-1)}$ 且整个布局 $\boldsymbol{L}$ 中没有介于 g_{si} 和 $g_{s(i-1)}$ 之间的元素, 则称这两个元素是自上而下连续递增的. 如果某元素可以移除, 则从该元素开始自上而下连续递增的几个 (不大于 M) 元素可以一次性移除. 将栈中所有自上而下连读递增的元素分为一段, 设栈 s 被分成的段数为 n_s^{SEG}, 段 j $(j = 1, 2, \cdots, n_s^{\mathrm{SEG}})$ 中的元素个数记为 λ_{js}.

定理 1 以布局 $\boldsymbol{L}$ 表达的板坯出库问题总移动次数的下界为

$$\ell = \sum_{s=0}^{S-1} \sum_{j=1}^{n_s^{\mathrm{SEG}}} \left\lceil \frac{\lambda_{js}}{M} \right\rceil, \tag{3}$$

其中 $\lceil \bullet \rceil$ 表示向上取整.

证明 因为板坯出库问题中一次可以移动的最大元素个数为 M, 所以 λ_{js} 个元素需要 $\lceil \lambda_{js}/M \rceil$ 次移动. 移动包括 λ_{js} 个元素的一段元素只有两种情形: 第 1 种情形是该段元素可以直接移除, 显然移除步数为 $\lceil \lambda_{js}/M \rceil$. 第 2 种情形是该段元素不可直接移除, 必须首先进行一次或多次移位, 而后移除. 在第 2 种情形下又有两种子情形: 第 1 种子情形, 若该段元素在最终移除之前每次移位时都没有和目的栈的元素合并成新的段, 则最终移除该段元素的步数为 $\lceil \lambda_{js}/M \rceil$, 总移动 (含移位) 步数大于 $\lceil \lambda_{js}/M \rceil$; 第 2 种子情形, 若该段元素在最终移除之前移位时和目的栈的元素合并为新的段, 则可与其他元素合并移除. 但是, 注意

到每一次移位至多合并两个段, 即源栈和目的栈中的段数总和至多减少 1. 也即移除步数至多减少 1; 因此, 该子情形下若要减少一次移除, 则必须以至少增加一次移位为代价. 综上, 对所有栈的所有段按 $\lceil \lambda_{js}/M \rceil$ 求和是板坯出库问题的一个下界. □

例如, 式 (2) 描述的板坯出库问题的下界为 $\ell = 2+3+3+2 = 10$.

3 树搜索算法

受文献 [13] 的启发, 本文提出了一个树搜索算法来求解板坯出库问题. 基本思想为: 首先利用贪婪算法给出问题的一个初始解作为历史最优解, 然后采用分枝定界过程求解. 对于初始布局, 生成若干复合移动, 每个复合移动均由一系列移动构成. 对当前布局按每个复合移动进行分枝, 如果满足终止条件, 则结束; 如果找到一个完整的解, 则根据需要更新历史最优解; 如果当前布局应用该复合移动后的布局仍非空, 即在该分枝未找到完整的解, 则对新的布局继续生成若干复合移动, 继续搜索.

在介绍算法的具体过程之前, 给出如下定义.

定义 1　如果一个移位同时满足如下两个条件: 1) 源栈中的最小元素不在栈顶, 移位的元素在该最小元素之上; 2) 目的栈为空或其中所有元素均大于移位的元素. 则称其类型为 I.

定义 2　如果一个移位满足如下条件: 源栈中含有整个布局中的最小元素, 移位的元素在该最小元素之上. 则称其类型为 II.

定义 3　如果一个移位同时满足如下条件: 1) 源栈中的最小元素在栈顶, 移位的元素自上而下递增; 2) 目的栈为空或其中所有元素均大于移位的元素. 则称其类型为 III.

根据上述定义, 类型为 I 的移位中移动的元素在源栈中堆放在需要先移除的元素之上, 而在目的栈中则不然; 类型为 II 的移位中移动的元素在源栈中堆放在所有栈中需要最先移除的元素之上; 类型为 III 的移位中移动的元素在源栈和目的栈中都是需要首先移除的.

此外, 定义 $m_1 + m_2$ 表示将移动 (或移动序列) m_2 追加到移动 (或移动序列) m_1 之后, $\boldsymbol{L} \oplus m$ 表示将移动 (或移动序列) m 应用于布局 $\boldsymbol{L}$.

3.1 贪婪算法

贪婪算法用来快速得到板坯出库问题的一个可行解, 包括以下主要步骤.

Step 1: 令 $\boldsymbol{L}$ 为板坯出库问题的初始布局, s^{OPT} 为空.

Step 2: 如果 $\boldsymbol{L}$ 有可行的移除, 则令该移除为 m, 转 Step 5.

Step 3: 如果 $\boldsymbol{L}$ 有可行的类型为 I 的移位, 则令该移位 (如果类型为 I 的移位多于 1 个, 则取其中移动元素数目最大的) 为 m, 转 Step 5.

Step 4: 如果 $\boldsymbol{L}$ 有类型为 II 的移位, 则令该移位 (如果类型为 II 的移位多于 1 个, 则取其中移动元素数目最大的) 为 m, 转 Step 5; 否则, 该板坯出库问题无可行解, 算法终止.

Step 5: $s^{\mathrm{OPT}} = s^{\mathrm{OPT}} + m$, $\boldsymbol{L} = \boldsymbol{L} \oplus m$. 如果新的布局 $\boldsymbol{L}$ 非空, 则转 Step 2; 否则, s^{OPT} 是得到的可行解, 算法结束.

3.2 执行树搜索

树搜索算法为一个递归算法, 其参数为布局 $\boldsymbol{L}$ 和一个不完整解 s^{CUR}, 初始调用该算法时 $\boldsymbol{L}$ 为板坯出库问题的初始布局, s^{CUR} 为空. 树搜索算法的主要步骤如下.

Step 1: 如果算法的运行时间已经达到预定的搜索时间, 则算法结束.

Step 2: 如果 s^{OPT} 中的移动次数等于初始出库问题的下界, s^{OPT} 为最优解, 则算法结束.

Step 3: $\boldsymbol{L}$ 的若干复合移动, 记为 Θ, 具体算法见 3.3 节.

Step 4: 对 Θ 中的复合移动按复合移动 m 中包括的移动次数与 $\boldsymbol{L} \oplus m$ 的下界之和从小到大的顺序进行排列. 如果 Θ 中复合移动的数目大于预先设定的每节点最大分枝数 n^{BRANCH}, 则保留 Θ 中的前 n^{BRANCH} 个复合移动. 令 m 为 Θ 中的第 1 个复合移动.

Step 5: 如果 $\boldsymbol{L} \oplus m$ 为空布局, 则 $s^{\mathrm{CUR}} + m$ 为一个完整解, 更新 s^{OPT}(若 $s^{\mathrm{CUR}} + m$ 优于 s^{OPT}, 则令 $s^{\mathrm{OPT}} = s^{\mathrm{CUR}} + m$), 转 Step 7.

Step 6: $\boldsymbol{L}' = \boldsymbol{L} \oplus m$, $s^{\mathrm{CUR}'} = s^{\mathrm{CUR}} + m$, 以 $\boldsymbol{L}'$ 和 $s^{\mathrm{CUR}'}$ 为参数调用该算法继续执行树搜索.

Step 7: 如果 m 不是 Θ 中的最后一个复合移动, 则令 m 为 Θ 中的下一个复合移动, 转 Step 5.

3.3 生成复合移动

生成复合移动的算法也是一个递归算法, 主要参数为: 当前布局 $\boldsymbol{L}$, 移动序列 m^{CUR}, 一个用来控制复合移动分枝规模的参数 n^{CM} 和复合移动序列 Θ. 当树搜索算法调用生成复合移动的算法时, $\boldsymbol{L}$ 为当前布局, $m^{\mathrm{CUR}} = \Phi$(为空), $n^{\mathrm{CM}} = 1$, $\Theta = \Phi$. 生成复合移动算法的主要步骤如下.

Step 1: 如果算法的运行时间已经达到预定的搜索时间, 则算法返回.

Step 2: 生成 $\boldsymbol{L}$ 的若干“有意义的”移动, 令 m 为

其中第 1 个有意义的移动, 具体算法见 3.4 节.

Step 3: 如果 $\boldsymbol{L}\oplus m$ 为一个空布局, 则得到一个复合移动 $m^{\rm CUR}+m$, 将该复合移动加入 Θ, 转 Step 7.

Step 4: 如果 $\boldsymbol{L}\oplus m$ 的下界表明该移动已经没有搜索价值, 则转 Step 7.

Step 5: 若 $n^{\rm CM}$不小于预先设定的阈值 $n^{\rm CM\_MAX}$, 则将得到的复合移动 $m^{\rm CUR}+m$ 加入 Θ, 转 Step 7.

Step 6: 以 $\boldsymbol{L}\oplus m$, $m^{\rm CUR}+m$, $n^{\rm CM}$ 与本次得到的有意义的移动个数的乘积和 Θ 为参数, 调用该算法继续生成复合移动.

Step 7: 如果 m 不是最后一个有意义的移动, 则令 m 为下一个有意义的移动, 转 Step 3.

3.4 生成“有意义的”移动

生成一个布局 $\boldsymbol{L}$ 的有意义的移动的算法步骤如下.

Step 1: 如果 $\boldsymbol{L}$ 有可行的移除, 则返回该移除, 算法结束.

Step 2: 如果 $\boldsymbol{L}$ 有类型为 I 的移位, 则返回所有类型为 I 的移位, 算法结束.

Step 3: 返回若干类型为 II 或 III 的移位, 其中返回的类型为 II 和 III 的移位的最大数目分别不超过预先设定的阈值 $n^{\rm II\_MAX}$ 和 $n^{\rm III\_MAX}$.

4 实验与分析

利用 C++ 语言在微软 Visual Studio 2008 环境下实现了本文提出的树搜索算法, 在 Intel(R)Core(TM) 2 Quad CPU Q9400 @2.66 GHz, @2.66 GHz 3.49 GB 内存的计算机上运行, 通过大量随机生成的算例对算法进行了测试.

在所有算例中, 参照文献 [13] 的参数设置, 设定: $n^{\rm II\_MAX}=3$, $n^{\rm III\_MAX}=2$, $n^{\rm CM\_MAX}=10$, $n^{\rm BRANCH}=5$. 另外, 根据某钢铁企业热轧板坯库所配置的吊机的规格, 设定 $M=3$, 根据实践需要, 设定最长搜索时间为 30 s.

首先, 随机生成一个小规模算例来说明算法逻辑上的合理性. 其中, $S=6$, $H=6$, 随机生成初始布局

$$\boldsymbol{L}_1=\begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\17&7&3&14&13&1\\4&9&6&12&2&11\\10&18&15&16&5&8\end{bmatrix},$$

其下界为 18, 贪婪算法给出的初始解长度为 23, 算法运行时间小于 1 s, 得到的最好解 (长度为 22) 为

$$\begin{aligned}s^{\rm OPT}=\,&5\xrightarrow{1}-1,5\xrightarrow{1}3,2\xrightarrow{2}1,4\xrightarrow{1}2,4\xrightarrow{1}-1,\\&1\xrightarrow{1}-1,4\xrightarrow{1}1,0\xrightarrow{1}4,0\xrightarrow{1}-1,1\xrightarrow{3}-1,\\&5\xrightarrow{1}-1,1\xrightarrow{1}-1,0\xrightarrow{1}-1,3\xrightarrow{1}-1,3\xrightarrow{1}5,\\&3\xrightarrow{1}-1,2\xrightarrow{1}-1,5\xrightarrow{1}-1,2\xrightarrow{1}-1,3\xrightarrow{1}-1,\\&4\xrightarrow{1}-1,1\xrightarrow{1}-1.\end{aligned}$$

由分析可知, 初始布局中编号最小的元素 1 位于栈 5 的顶部, 可以直接移除, 对应解 $s^{\rm OPT}$ 的第 1 步 $5\xrightarrow{1}-1$. 尽管首先将元素 13 移到其他栈, 并将元素 1 移到栈 4, 可将元素 1 和 2 同时移除, 但因为不能节省总的移动次数, 所以没出现在 $s^{\rm OPT}$ 中, 这与证明定理 1 时的分析一致. 移除元素 1 之后, $s^{\rm OPT}$ 将元素 11 移到栈 3, 对应第 2 步 $5\xrightarrow{1}3$. 元素 11 在移位之前压在需先移除的元素 8 之上, 而目的栈中所有元素均大于 11, 所以不受元素 11 的影响, 可见该移位 $5\xrightarrow{1}3$ 的类型为 I. 接下来, 移位 $2\xrightarrow{2}1$ 将元素 3 和 6 同时移到栈 1, 可使栈 2 只剩下一个元素 15, 从而可接受编号更大的元素而不必再次移位, 分析可知其类型为 III. 进而, 类型为 I 的移位 $4\xrightarrow{1}2$ 将元素 13 移至栈 2, 而 $4\xrightarrow{1}-1$ 将元素 2 移除, $1\xrightarrow{1}-1$ 将元素 3 移除, 布局变为

$$\boldsymbol{L}_1'=\begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&0\\0&6&0&11&0&0\\17&7&0&14&0&0\\4&9&13&12&0&0\\10&18&15&16&5&8\end{bmatrix}.$$

此时, 元素 4 是整个布局中最小的, 需首先移除, 要移除元素 4 必须首先移动上面的元素 17; 然而其他所有栈中都有小于 17 的元素, 即不论将该元素移到哪个栈, 都将压住下面的元素, 从而导致进一步移位. $s^{\rm OPT}$ 中 $4\xrightarrow{1}1$ 首先将元素 5 移到栈 1 (以后移除时可与元素 6, 7 合并), 此时栈 4 为空; 而后将元素 17 移到栈 4, 由分析可知该移位 $0\xrightarrow{1}4$ 的类型为 II; 接下来一系列的移除 $0\xrightarrow{1}-1,1\xrightarrow{3}-1$,5 $\xrightarrow{1}-1,1\xrightarrow{1}-1,0\xrightarrow{1}-1,3\xrightarrow{1}-1$ 将元素 4 $\sim$11 移除; 后续的操作原理上类似, 在此不作详细讨论. 以上分析表明该树搜索算法的合理性.

根据生产实际随机生成了大量的算例, 进一步测试了算法的有效性. 根据某钢铁企业热轧板坯库的实际情形, 设 $H=12$, 垛位堆满率取 0.5, 初始时各垛堆放高度为 $12\times0.5=6$, 设栈的数目分别为 8 和 10, 分别随机生成了 10 个算例, 算法很好地求得了满意解, 其中下界、历史最优值等信息如表 1 所示. 在这两组算例中, 板坯出库问题的下界十分接近板坯数目, 这是因为随机生成的算例中“自上而下连续递增”的元素出现的概率很小, 即绝大多数情形下每个元素单独构成一段. 此外, 各算例中贪婪算法都给出了一个较

好的解, 而对于多数算例在 30 s 的搜索时间内解的质量都得到了较好的提高. 上述计算性能能够满足实际生产的需要, 可以很好地提高生产效率. 此外, 实际生产中的板坯并不是随机堆放的, 而是在堆垛时就已经充分考虑了出库顺序, 即实际生产中的板坯比随机生成的算例中堆放得更加有序, 因此实际生产中搜索的效果将更好.

表 1　实际规模算例的求解情况

序号	S	元素数目	ℓ	初始目标值	最优目标值
1	8	48	47	76	67
2	8	48	47	81	78
3	8	48	47	79	72
4	8	48	47	71	64
5	8	48	47	79	66
6	8	48	47	80	72
7	8	48	46	71	70
8	8	48	48	81	69
9	8	48	48	82	80
10	8	48	45	79	70
11	10	60	58	93	88
12	10	60	60	87	80
13	10	60	60	105	92
14	10	60	59	97	87
15	10	60	60	99	86
16	10	60	58	89	89
17	10	60	60	101	98
18	10	60	58	101	89
19	10	60	60	97	97
20	10	60	59	91	87

此外, 对贪婪算法的性能进行了一些极限测试. 设 $S = 10$, $H = 12$, 各栈的初始堆放高度由 7 逐渐增加至 11, 随机生成了 10 个算例 (每一个初始堆放高度对应 2 个算例), 该贪婪算法都成功地求得了算例的一个效果不错的可行解. 其中一个算例 (算例 30) 的初始布局为

$$\boldsymbol{L}_{30} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 27 & 46 & 79 & 44 & 81 & 62 & 33 & 106 & 102 & 21 \\ 56 & 8 & 10 & 40 & 74 & 17 & 105 & 66 & 52 & 90 \\ 77 & 85 & 31 & 35 & 103 & 36 & 13 & 108 & 63 & 76 \\ 11 & 22 & 97 & 28 & 20 & 42 & 54 & 98 & 67 & 1 \\ 68 & 23 & 9 & 15 & 82 & 16 & 92 & 49 & 55 & 18 \\ 73 & 24 & 84 & 19 & 32 & 61 & 78 & 91 & 75 & 89 \\ 70 & 86 & 50 & 60 & 30 & 3 & 37 & 100 & 65 & 109 \\ 88 & 25 & 14 & 26 & 95 & 107 & 83 & 94 & 38 & 93 \\ 45 & 5 & 47 & 71 & 12 & 2 & 4 & 53 & 99 & 80 \\ 96 & 110 & 51 & 39 & 48 & 69 & 41 & 43 & 72 & 58 \\ 6 & 7 & 87 & 64 & 29 & 101 & 104 & 57 & 59 & 34 \end{bmatrix}.$$

算法给出的初始解的长度为 238, 相应的解略.

5　结　论

热轧板坯库的高效管理对连铸-热轧作业至关重要, 其出库问题是一个复杂的组合优化问题; 然而, 在公开发表的文献中还没见到对该问题的研究. 本文以一次可以移动多块的板坯出库问题为研究对象, 以极小化总移动次数为优化目标, 根据分段的思想提出了问题的一个下界, 开发了一个树搜索算法. 该算法从一个贪婪搜索得到的初始解开始, 以复合移动的方式对“有意义的”移动进行递归搜索, 直至得到最优解或达到预定搜索时间. 大量的实验验证了本文算法的合理性, 理论分析表明该算法能够在较短的时间内得到一个满意解, 特别是其中的贪婪算法能够快速生成板坯出库问题的高质量的可行解. 此外, 理论分析还表明, 该算法在实际生产中将具有更好的效果, 并体现出该算法的理论和应用价值.

参考文献(References)

[1] 卢克斌, 黄可为, 汪定伟, 等. 钢铁企业合同计划与余材匹配的集成优化方法[J]. 控制与决策, 2009, 24(1): 71-75.
(Lu K B, Huang K W, Wang D W, et al. Integrated optimization approach of contract planning and surplus inventory matching of steel mill[J]. Control and Decision, 2009, 24(1): 71-75.)

[2] 卢克斌, 黄可为, 张瑞友, 等. 炼钢-连铸作业计划的基于仿真的遗传算法[J]. 控制与决策, 2011, 26(1): 137-140.
(Lu K B, Huang K W, Zhang R Y, et al. Simulation based genetic algorithm for production planning of steelmaking-continuous casting process[J]. Control and Decision, 2011, 26(1): 137-140.)

[3] 谭园园, 宋建海, 刘士新. 加热炉优化调度模型及算法研究[J]. 控制理论与应用, 2011, 28(11): 1549-1557.
(Tan Y Y, Song J H, Liu S X. Model and algorithm for scheduling reheating furnace[J]. Control Theory & Applications, 2011, 28(11): 1549-1557.)

[4] 谭园园, 宋建海, 刘士新. 加工时间可控的炼钢调度问题两阶段模型及优化算法[J]. 控制理论与应用, 2012, 29(6): 696-707.
(Tan Y Y, Song J H, Liu S X. A hybrid two-phase algorithm and mathematical model for steelmaking and continuous casting with controllable processing time[J]. Control Theory & Applications, 2012, 29(6): 696-707.)

[5] 唐立新, 杨自厚. 热轧实施计划中最优倒垛问题的整数规划模型及遗传算法[J]. 自动化学报, 2000, 26(4): 461-469.
(Tang L X, Yang Z H. Integer programming model and modified genetic algorithm for optimal turned-out slab pile for hot rolling schedule[J]. Acta Automatica Sinica, 2000, 26(4): 461-469.)

[6] 唐立新, 杨自厚, 胡国奋. 板坯最优倒垛问题的有效启发式算法[J]. 系统工程学报, 2001, 16(2): 121-127.

(Tang L X, Yang Z H, Hu G F. Effective heuristic algorithm for optimal turned-out slab pile problem[J]. J of Systems Engineering, 2001, 16(2): 121-127.)

[7] Tang L, Ren H. Modelling and a segmented dynamic programming-based heuristic approach for the slab stack shuffling problem[J]. Computers and Operations Research, 2010, 37(2): 368-375.

[8] 任会之, 唐立新. 考虑库区吊机能力的板坯倒垛问题的建模与优化方法研究[J]. 自动化学报, 2010, 36(4): 586-592.
(Ren H Z, Tang L X. Study on modeling and optimization method for the slab stack shuffling problem considering area crane capacity[J]. Acta Automatica Sinica, 2010, 36(4): 586-592.)

[9] Singh K A, Srinivas, Tiwari M K. Modelling the slab stack shuffling problem in developing steel rolling schedules and its solution using improved Parallel Genetic Algorithms[J]. Int J of Production Economics, 2004, 91(2): 135-147.

[10] Kim K H, Hong G P. A heuristic rule for relocating blocks[J]. Computers and Operations Research, 2006, 33(4): 940-954.

[11] Caserta M, Voβ S. A cooperative strategy for guiding the corridor method[J]. Studies in Computational Intelligence, 2009, 236(1): 273-286.

[12] Caserta M, Voβ S, Sniedovich M. Applying the corridor method to a blocks relocation problem[J]. OR Spectrum, 2011, 33(4): 915-929.

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

[14] Vis I F A, Roodbergen K J. Scheduling of container storage and retrieval[J]. Operations Research, 2009, 57(2): 456-467.

[15] Lee Y, Lee Y J. A heuristic for retrieving containers from a yard[J]. Computers and Operations Research, 2010, 37(6): 1139-1147.

[16] 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.

[17] 李浩渊. 集装箱码头物流系统的基于仿真的优化方法研究[D]. 沈阳: 东北大学 信息科学与工程学院, 2010.
(Li H Y. Research on simulation based optimization approaches for logistics systems in container port[D]. Shenyang: College of Information Science and Engineering，Northeastern University, 2010.)

下 期 要 目

第 33 届中国控制会议征文通知

中国控制会议是由中国自动化学会控制理论专业委员会发起的系列学术会议, 现已发展成为控制理论与技术领域的国际性学术年会. 第 33 届中国控制会议由中国自动化学会控制理论专业委员会和中国系统工程学会共同主办, 南京理工大学承办, 将于 2014 年 7 月 28-30 日在享誉世界的古都南京举行. 有关会议的具体事宜请登录会议网站进行了解.

会议网站: http://ccc.amss.ac.cn/2014/cn/，http://ccc2014.njust.edu.cn