分类号_____ 学号 M201672835

学校代码 10487 密级_____

华中科技大学

硕士学位论文

启发式算法求解二维矩形切割优化问题研究

学位申请人： 黄丹妮

学 科 专 业： 计算机软件与理论

指 导 教 师： 吕志鹏 教授

答 辩 日 期： 2019 年 5 月

A Dissertation Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering

A Heuristic Algorithm for Two-Dimensional Rectangle Cutting Optimization

Candidate : Huang Danni

Major : Computer Software & Theory

Supervisor : Prof. Lü Zhipeng

Huazhong University of Science & Technology

Wuhan 430074, P. R. China

May, 2019

摘 要

二维矩形切割优化问题属于组合优化问题的范畴，已被证明为 NP 难问题。在化工业生产中，常常会涉及到原料或半成品的切割等工序。而二维矩形切割问题相对于单维和多维切割问题在工业界的应用最为广泛，有着诱人的发展前景。因此研究该问题对于提升企业经济效益和推动学术界理论研究都有着重要的价值。

二维矩形切割优化问题是将标准尺寸的原料按照 Guillotine 切割方式切割成特定尺寸的成品，在此过程中会产生部分废料，优化的目标是最小化原料的总用量和最小化切割产生的废料，为此本文设计了求解该问题的一种迭代的带扰动机制和分支定界的启发式树搜索算法（IHTS-P），通过深度优先搜索探索解空间，加入分支策略对解空间进行细分，而剪枝策略则是对无意义的解空间进行剪枝便于算法加速，为了提高求解优度和求解效率，本文还创新性地提出了迭代优化策略和禁忌扰动机制：通过迭代优化中每一轮的集中性的搜索来改善解空间，通过禁忌扰动机制来增强解的疏散性。

利用源自工业界的多种真实算例对 IHTS-P 算法进行测试，并和本文提出的两种混合整数规划模型：完整模型（CM）、迭代模型（IM）测试结果进行对比，同时和最优解进行对比，以及在不同的运行时间下对比各算法的求解效果，结果表明无论是求解性能还是求解效率上，IHTS-P 算法都有较大优势。

关键词： NP 难问题；切割问题；启发式算法；树搜索算法；迭代优化；扰动机制

Abstract

The Two-Dimensional Rectangular Cutting Optimization Problem is a combinatorial optimization problems and it has been proved to be an NP-hard problem in computational complexity theory. In the chemical industry production, processes such as cutting of raw materials or semi-finished products are often involved. The Two-Dimensional Rectangular Cutting Problem is the most widely used in industry compared to single-dimensional and multi-dimensional cutting problems, and it has considerable development prospects. Therefore, studying this problem has great importance and significance to improve the economic effect of enterprises and promote theoretical research in academia.

The Two-Dimensional Rectangular Cutting Optimization Problem is to cut a standard size raw material into a specific size through the Guillotine cut, in which process the waste is generated. The objective of the problem is to minimize the given ordered raw materials and the geometrical loss of the cutting patters applied on materials. The Iterative Perturbation mechanism and a branch and bound Heuristic Tree Search algorithm (IHTS-P) for solving this problem has been proposed. The depth-first search is used to explore the solution space, and the branch strategy is added to subdivide the solution space. In addition, the pruning strategy is to speed up the algorithm. In order to enhance the quality and efficiency of the solution, an iterative optimization strategy and a tabu-based perturbation one have been employed. The technique has achieved the balance of determination and diversification.

In order to evaluate the performance of IHTS-P, a set of real-world instances are tested. From the empirical experimental evaluation, it is shown that this method is able to gain significant better solution than the two mixed integer programming models proposed in this paper: Complete Model (CM) and Iterative Model (IM). Furthermore, the experiments not only illustrate that the proposed method performs remarkable with regard to the current best solutions, but also analyze the solutions of each algorithm on different running times.

Key words： NP-hard problem; cutting problem; heuristic algorithm; tree search algorithm; iterative optimization; perturbation mechanism

目 录

1 绪论

1.1 课题背景和意义

进入21世纪，随着以深度学习为代表的人工智能算法的成功应用，以及大数据、云计算等新兴技术的日趋成熟，人工智能迎来了第三次发展热潮。包括我国在内的世界各科技强国相继将人工智能技术的研发提升到了关系国家发展的战略高度。“互联网+”与“中国制造2025”等科技强国战略部署，都离不开人工智能的支持，而智能化程度也成为了评价发展成果的重要指标。近二十年来，我国的信息产业不断发展，逐步向万物智慧互联推进。光纤通信与 4G、5G 移动网络等基础网络设施的建设为大数据的实时快速传输铺平了道路。以物联网为代表的信息化建设实现了数字化，从各个角落采集了海量数据，为统一管理与统筹调度打下了坚实的基础。以深度学习为代表的数据挖掘与统计学习方法中提出的神经网络实现了对海量数据进行深度的提炼和准确的分析，并能够对未来的趋势进行可靠的预测，辅助做出单步决策。因此在完整的实时信息和精准的分析预测等先决条件已经准备就绪的情况下，怎样根据已知信息做出合理的决策，以期在复杂的生产和运营的各个环节，达到深度的全局优化的效果，这就是运筹学要解决的问题了。因此，未来运筹学与人工智能相结合，将成为必然的趋势。

运筹学常用于解决现实生活中的复杂问题。在有限个对象集合中找出最优对象的一类优化问题称为组合优化问题，它是运筹学中的一个重要分支。组合优化问题算法就是在离散变量的情况下，从有限多个解中快速寻求极大解或极小解的方法。常见的组合优化问题有：旅行商问题（Traveling Sales Problem，TSP）[1]，车辆路径问题（vehicle Routing problem，VRP）[2]，装箱问题（Packing Problem）[3]和布尔可满足性问题（Boolean Satisfiability Problem，SAT）[4]等。

切割优化问题（Cutting Optimization Problem）[5]属于组合优化问题的范畴。在化工业生产中，常常会涉及到原料或半成品的切割等工序。切割问题是将标准尺寸的原料（例如纸卷、玻璃或金属板）切割成特定尺寸的碎料，在此过程中会产生部分余料。

此时切割问题的优化就显得尤为重要，优化的目的是最大程度的降低切割产生的残料，来达到降低原材料的消耗。常见的应用场景如：切割管道，电缆或者钢筋；玻璃行业中，需要将玻璃切割成指定大小的矩形；纸张、薄膜；金属行业等需要以大型卷材生产并进一步切割成较小的单元。

上述与实际应用相联系的切割问题可以通过切割的维数进行分类：将切割管道或条状的原材料的问题划分为一维切割问题；将玻璃等生产中涉及到的切割问题划分为二维切割问题；涉及到三维切割问题的应用并不多，比较典型的是装箱问题[6]，例如将物体打包到集装箱中。本文主要研究的是二维矩形切割问题的优化。由于切割的顺序和切割的尺寸都会导致最终切割方案的不同，因此任何一种可能的切割方案都是问题的解。考虑到随着原料尺寸的不断增大、切割尺寸的种类扩大，切割方案的数目会随之指数级增长，因此该问题已被证明为 NP 难问题。

经典的 NP 难问题有：TSP 问题[1]、背包问题[7]和最大团问题[8]等。一般来说，求解组合优化问题的方法有：完备性算法和非完备性算法两类。其中完备性算法是指只要给定的搜索时间足够长，算法一定能够搜索完所有的情况，并最终给出最优解。而非完备性算法则是即使给定足够的搜索时间，也不一定能够搜索完所有的分支，找到最优解。完备性算法多以树搜索为框架，以分支限界方法（Branch and Bound，BB）[9]为基础，常常使用数学规划的方法，在算例规模较少的情况下，算法的效率很高，能够求出最优解，然而对于复杂的大规模算例则效果不太理想。非完备性算法则使用近似的方法来解决优化问题。以局部搜索（Local Search）[10]方法为例，该方法能够快速找到一个有质量保证的解，然后对于解进行调整不断的找到更好的解，因此搜索路径具有随机性，无法搜索完所有解空间。由于该方法不需要遍历所有解空间，所以对于大规模算例的效果一般要高于完备性算法。然而，单纯的使用各种经典的算法依然无法满足实际的工业生产生活问题的求解要求。启发式算法不同于纯粹的数学推导，它无法用数学语言来证明解的最优性，甚至在某些情况下无法说明和最优解的近似程度，但其能在性能和优度之间达到很好的平衡。对于一些完备性算法的分支变元的选择问题，往往加入启发式方法，能够使得算法性能大幅提升。对于以局部搜索算法为代表的非完备性算法，如果使用一些启发式的策略（如贪心算法[11]，禁忌算法

[12]，蚁群算法[13]，松弛[14]，惰性约束等）一般来讲可以在实际工业生产问题中找到一个近似最优解。

因此，对于切割优化问题的研究，是提升企业经济效益、合理利用有限资源、可持续发展的大势所趋。在实际应用场景中，切割优化问题相对于经典问题规模更加庞大，问题也更加复杂；同时在学术上具有挑战性，由于切割优化问题属于 NP 难问题，在研究过程中需要考虑选取何种算法，同时需要根据具体问题设计不同的算法，对研究人员提出了更高的要求。总而言之，切割优化问题是关乎加工型企业发展和提高企业经济效益的瓶颈问题，因此研究启发式算法求解二维矩形切割优化问题（Two Dimensional Rectangle Cutting Optimization，TDRCO）具有重大的实际和理论意义，值得深入研究。

1.2 国内外研究概况

学术界，对于切割优化问题的研究主要围绕二维切割问题。一方面二维切割问题更容易得到合理的切割方案，好的二维切割算法可以快速带来极大的经济效益；另一方面，二维切割问题应用的领域更加广泛，如钢铁、木材、玻璃、皮革、矿产、石料等制造业，研究二维切割问题的优化显得尤为迫切。切割问题（Cutting Stock Problem，CSP）最早由 Kantorovich 于 1939 年首次提出[5]，又称为装箱问题（Bin Packing Problem）[15]、分类问题（ Assortment Problem）[16]等，这些问题在本质上并没有太大区别，所以在此归为一类分析。在 1951 年，计算机普及之前，Kantorovich 和 Zalgaller 又建议用线性规划规划的方法来解决该问题，该方法后来被命名为列生成法（ Column Generation Method）[17]，而后针对这一基本问题和相关变种发展出一系列形式化模型和求解算法，总体来说，这些算法可以分为两大类：精确算法、混合算法。其中精确算法多以分支限界方法为基础，可以借助于数学规划类求解软件，而混合算法是将启发式算法与精确算法结合在一起的一类算法。

- 求解二维矩形切割优化问题的精确算法

精确算法是指搜索整个解空间求得最优解的算法。只要给定的搜索时间足够长，算法一定能够搜索完所有的情况，并最终给出最优解。目前提出的精确算法有很多，

主要有分支定界法[9]、整数规划算法[18]、动态规划算法[19]和割平面法（Cutting Plane Method）[20]。

Yanasse [21]在文献中提出用整数线性规划的方法来解决含约束和不含约束的二维切割模型，这些模型能够有效地求解中小规模的算例。在求解无约束的二维切割问题上， Silva [22]受 Dyckhoff 一维切割模型[23]的启发，提出基于二维切割模型的整数规划模型，考虑了两种切割模式，同时解决了物品旋转，切口长度和剩余块的价值等问题。Furini [24]建立了二维切割问题中 Guillotine 切割方式（在板材上的每次切割轨迹是一条边到边的连通直线）下的混合整数线性模型并结合 CPLEX 求解器，可以求解具有较大规模的算例。

Russo 等[25]针对大型算例的求解过程提出了减少搜索空间和避免冗余的改进的动态规划算法。Hifi [26]等提出了求解双约束二维切割问题的近似算法和精确算法，其中精确算法采用分支限界法，以近似算法得到的下界作为起始，通过求解松弛的背包问题来获得特定的下界。Bekrar [27]采用了一种使用上下界和可行性测试的二分法算法来求解 Guillotine 切割方式下的二维切割问题。

- 求解二维矩形切割优化问题的启发式算法

当传统的算法无法求解规模较大的算例时，启发式算法应运而生。启发式算法旨在通过牺牲最优性，准确性，精度或完整性以获得比传统方法更快更有效的方式解决问题。启发式算法常用于解决 NP 完全问题中的一类决策问题。在这些问题中，没有已知的有效方法来快速准确地找到解决方案。启发式算法可以单独生成解决方案，也可以用于提供良好的基础，并辅以优化算法。它无法用数学语言来证明，但其能在性能和优度之间达到很好的平衡。

Byungsoo 等[28]针对玻璃切割中最小化废料的切割和调度问题，提出了启发式算法，并给出最坏情况下的性能界限。Ramon 等[29]提出了求解二维切割问题的几种启发式算法：它遵循 Gomory 的列生成方案[30]，在每次迭代中，他们分别采用动态规划、贪心自适应搜索算法（GRASP）和禁忌搜索（Tabu Search，TS）生成质量和时间要求不同的解决方案。Fabio [31]提出了基于列生成的启发式算法来解决二维切割问

题，定义了用于解决该问题的混合整数线性规划（Mixed Integer Linear Programming）模型，以及基于动态规划的启发式过程。Bernardo 等[32]针对玻璃生产过程中的切割问题采用变领域搜索算法（Variable Neighborhood Search，VNS）使得邻域大小显著减小，大幅提高了计算效率。Frederico [33]提出的变领域搜索策略应用于二维切割问题上，在求解中等规模的算例时，VNS 与动态规划相结合算法的结果明显优于构造启发式 VNS 算法。Tiwari [34]等针对 Guillotine 切割方式和非 Guillotine 切割方式下的二维切割问题采用了基于树编码的多目标遗传算法（Genetic Algorithm，GA）。将切割模式用二进制表示，并基于已有情况获得了全局最优解。Mohsen [35]讨论从包含多个瑕疵的矩形大物件中切割出给定类型的小矩形物品的问题时，提出一种基于动态规划的启发式方法。

Andreas 和 Tobias [36]提出了解决二维背包问题（Two Dimensional Knapsack Problem，2D-KP）的遗传算法，该问题中物体的放置类似于 Guillotine 切割方式，算法获得的解基于已有的方法具有一定的优势。Lodi [37]考虑了二维装箱问题的变体：物体满足 Guillotine 切割方式且不能旋转，提出了一种基于部分枚举的启发式算法。该算法可为大量类似问题提供已验证的最优解。基于 Fanslau 和 Bortfeldt [38]在容器装载问题上设计的树搜索算法（Tree Search Algorithm，TRSA），Andreas 和 Sabine [39]提出了一种改进的树搜索算法用于解决包含 Guillotine 切割方式约束的二维条带填充问题，它相比已有的算法具有一定的竞争力。Yaodong [40]等提出一种解决二维条带打包问题的启发式递归算法，它将递归结构和分支定界技术相结合，满足 Guillotine 约束。结果表明该算法比当前最近发布的几种算法更有优势。Krzysztof [41]提出了求解满足 Guillotine 约束的二维装箱问题的三种启发式算法，即首次拟合插入启发式算法（First-fit Insertion Heuristic），最佳拟合插入启发式算法（Best-fit Insertion Heuristic）和临界拟合插入启发式算法（Critical-fit Insertion Heuristic），并证明它们的算法效率和有效性。

1.3 本文主要工作及结构安排

本文主要研究的是 Guillotine 切割方式下的二维矩形切割优化问题。结合国内外

相关学者对此类问题的研究，通过对玻璃切割优化这一实际问题进行分析，了解玻璃切割的基本流程，理清了切割过程中的各类约束条件，明确了求解目标；构造了二维矩形切割的数学优化模型；采用启发式树搜索算法来探索解空间，包括分支策略和剪枝策略，并加入迭代优化和扰动机制对解进行优化；对求解结果进行评估、分析和调整。同时本文对我的研究成果进行了总结，并提出了今后完善的方向。

第一章详细介绍了本课题研究的背景和意义，并从求解二维矩形切割优化问题的精确算法和启发式算法研究现状两个方面介绍了国内外研究发展的现状。

第二章主要是二维矩形切割优化模型的建立，其中包括对问题的分析，约束条件、求解目标和决策变量的描述，并利用数学符号和数学公式，给出了两种数学模型：完整模型和迭代模型。

第三章我将详细描述本文中提出的启发式树搜索算法及相应的优化策略。首先介绍启发式树搜索算法的相关背景知识。接着依据算法的整体框架围绕：启发式树搜索、迭代优化和扰动机制三个方面进行介绍。给出启发式树搜索算法中的分支策略和剪枝策略的描述。同时算法中解的表示也给出了相应的说明。

第四章则是实验结果分析，对于第二章提出的两个数学模型，以及第三章提出的启发式树搜索算法，在测试算例集上分别测试求解性能，比较各数学模型求解结果的差异；对比启发式树搜索算法和数学模型的求解结果；将各算法与当前最优解进行比较；分析不同运行时间下的各算法的求解结果。对上述结果进行评价，验证该算法由理论到实际的可靠性。

第五章主要是对本论文研究工作的总结和评述，总结了本文中的重点工作、提出未来对系统中的不足将进行的完善工作。

2 切割优化模型实现

本章主要介绍二维矩形切割优化模型的建立，包括问题描述和问题建模。其中问题描述会介绍该问题的相应背景知识和问题中涉及到的相关概念解释和相应数据约定，问题建模主要介绍完整模型和迭代模型这两种数学模型。

2.1 问题概述

二维矩形切割优化问题属于组合优化问题和生产调度排班问题交叉的一个混合问题。考虑 Guillotine 切割方式为原料切割成成品的策略。目标是对于给定的一系列原料和每块原料对应的瑕疵图，以及需要切割的给定批次的成品，制定成品放置方案，设计成品切割顺序，遵守给定的约束，满足规定的切割任务要求，尽可能地减少切割过程中的原料损失。本文研究的二维矩形切割优化问题来源于工业界真实的案例，该问题与二维装箱问题的求解过程类似，但需要遵循一些额外的工业生产要求。下面给出玻璃切割详细的工作流程介绍和抽象出来的相关概念与数据约定。

2.1.1 问题来源

玻璃大多是通过一种称为“浮法”的工艺生产的。在这个过程中，首先将各种粉末（沙子和苏打水等）在大型熔炉内熔化在一起，形成一个液体玻璃带；然后用锡浴冷却固化；最后将得到的有色玻璃带切成大玻璃板（通常为 3m×6m，称为巨型玻璃）。之后将这些巨型玻璃通过滑板运输到变压器。一般来说，这些巨型玻璃并没有直接被使用，它们大部分都会根据客户的需求重新切割成较小的矩形件。这些较小的玻璃片根据切割图案切割成型，满足顾客或玻璃切割模式的约束条件，如 Guillotine 切割方式。切割方案可以看作是由各种尺寸的矩形块铺设的巨型铺板，其放置方式满足几何玻璃损失（剩余玻璃表面积太小而不能切割出新的块）尽可能低。

实际上，巨型玻璃在质量方面并不完美，可能是浮法制作过程中的固有缺陷。因此在完成浮法过程后，会用扫描仪为每块巨型玻璃建立瑕疵图（包含位置和规模等信息）。在本文研究中，这些瑕疵图可以认为是精确的，该信息存储在数据库中。在切割过程中，当瑕疵位于切割玻璃片中时，大多数情况下，这些玻璃片会因质量问题被

丢弃，需要从巨型玻璃中重新切割相同尺寸的玻璃片，这显着降低了生产线的生产率。为了避免这种损失，一种选择是使切割模式自适应测量出的瑕疵图，并将瑕疵定位在切割模式中的几何玻璃损失内。这就意味着需要最小化包含瑕疵的玻璃块的切割尺寸和数量，并最小化几何玻璃损失的尺寸。

2.1.2 相关概念与数据约定

- **Item：成品，订单，产品**

需要通过切割得到的小块成品玻璃；可以旋转 90 度，即交换宽度和高度值；每个成品一定属于某个有序的订单列表；属性：宽度值（总是不小于高度）、高度值。

- **Intermediate：中间产物**

切割过程中产生的包含成品，废料，余料的矩形区域；切割树的非叶子节点。

- **Quasi：半成品、准成品**

宽度相等且高度不小于成品高度的矩形区域。

- **Stack：有序订单列表**

一系列需要按顺序完成的切割订单；元素：订单。

- **Batch：订单列表集合**

一系列允许以任意顺序完成切割的有序订单列表；元素：有序订单列表。

- **Bin：原料**

待切割的大块玻璃；不允许旋转 (旋转无意义，切割方案等价)；属性：宽度值(总是不小于高度)、高度值、瑕疵信息。

- **Queue：有序原料列表**

一系列只能以给定顺序依次被切割的原料；元素：原料。

- **Defect：瑕疵**

原料上不允许出现在成品中或被切割轨迹穿过的区域；属性：左下角横坐标、左下角纵坐标、宽度值、高度值。

- **Waste：废料**

原料经过切割后剩下的不属于成品的部分。

- **Residual：余料**

每块原料执行第一刀切割后最右侧的一块不包含任何成品可重复使用的部分。

- **Guillotine 切割方式**

切割轨迹为若干条从一侧边缘到其对面边缘且垂直该边缘（平行的）的线段。换句话说，Guillotine（Edge to Edge）切割，如果应用于矩形板，则产生两个新的矩形板。这种类型的切割在切割玻璃的过程中是强制性的，否则会产生裂缝。如图 2-1 和图 2-2 分别描述了 Non-Gullotine 切割和 Guillotine 切割方式。

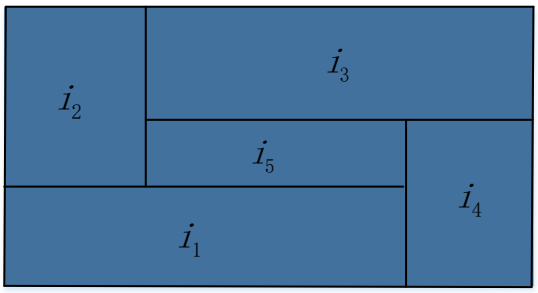


图 2-1 Non-Guillotine 切割

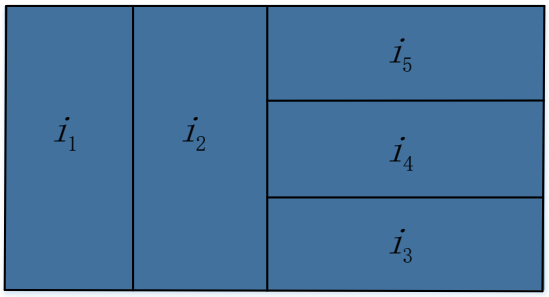


图 2-2 Guillotine 切割

- **切割模式（Cutting Pattern）**

由于技术限制，执行物品切割的机器的切割数量通常受限。任意成品只能最多经过三次切割得到其半成品；第一刀总是与高度方向平行；每一刀与前一刀的方向垂直。如图 2-3 描绘了一块包含瑕疵的原料在 Guillotine 切割约束下执行切割的方式。图 2-4 描述了两种有效的切割模式和一种无效的切割模式，切割模式中可以在三次切割后获得的半成品中进行至多一次的第四次切割。在有效的切割模式中第四次切割后只能生成一件成品和一件废料或两件成品，如图 2-4 (a)和(b)所示；而无效的切割模式中进行了两次的第四次切割，生成了两件成品和一件废料，如图 2-4 (c)所示，

这是切割模式中不允许的。

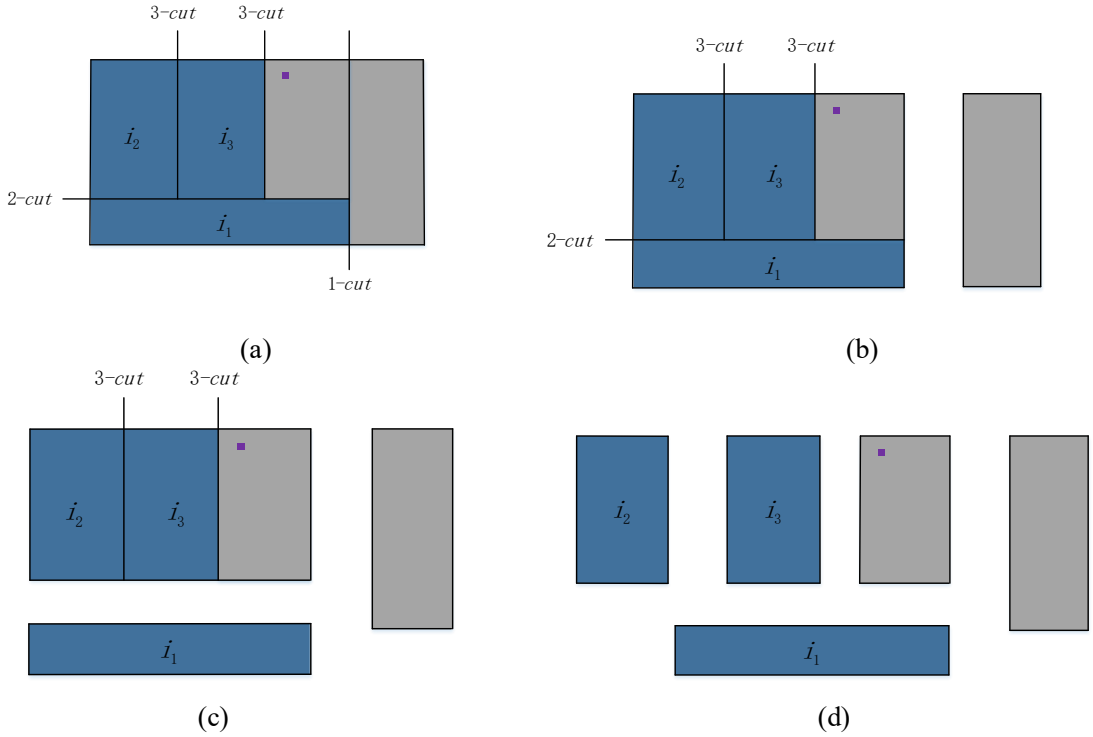


图 2-3 初始模式(a)及其执行切割后的变体(b-c-d)

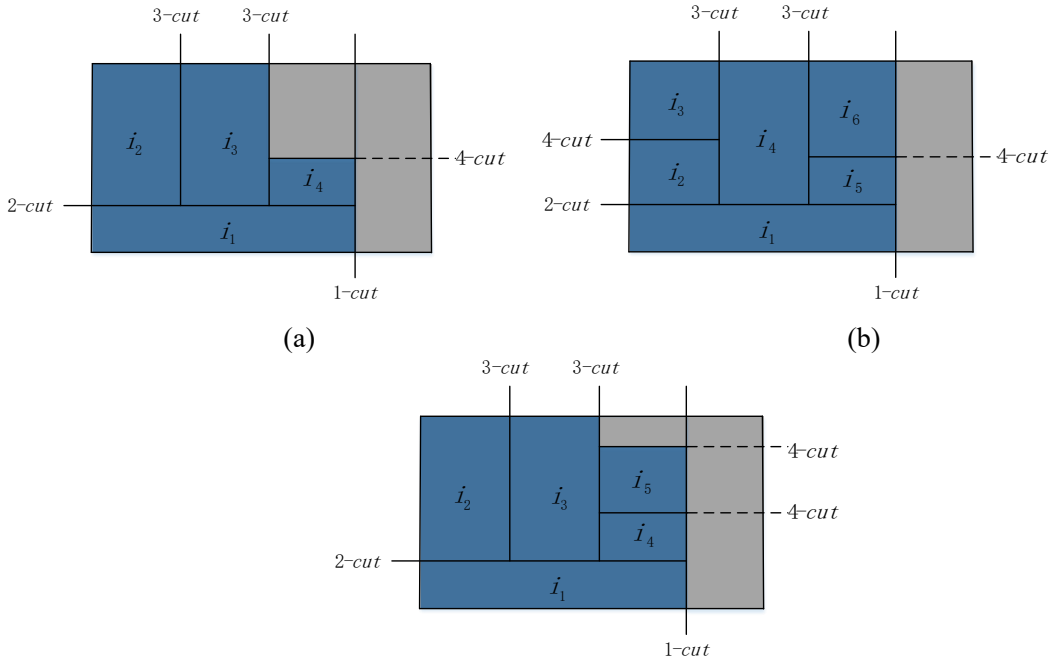


图 2-4 有效的切割模式(a-b)，无效的切割模式(c)

一块原料上成品的完工顺序可以通过深度优先切割树表示。它的根对应切割原料，其叶子结点对应切割的成品或废料或余料，对任意一个非叶子节点，其左子树上的所有成品的完工时间均早于其右子树上的所有成品。如图 2-5 是图 2-3 中成品完工顺序的深度优先切割树表示形式。

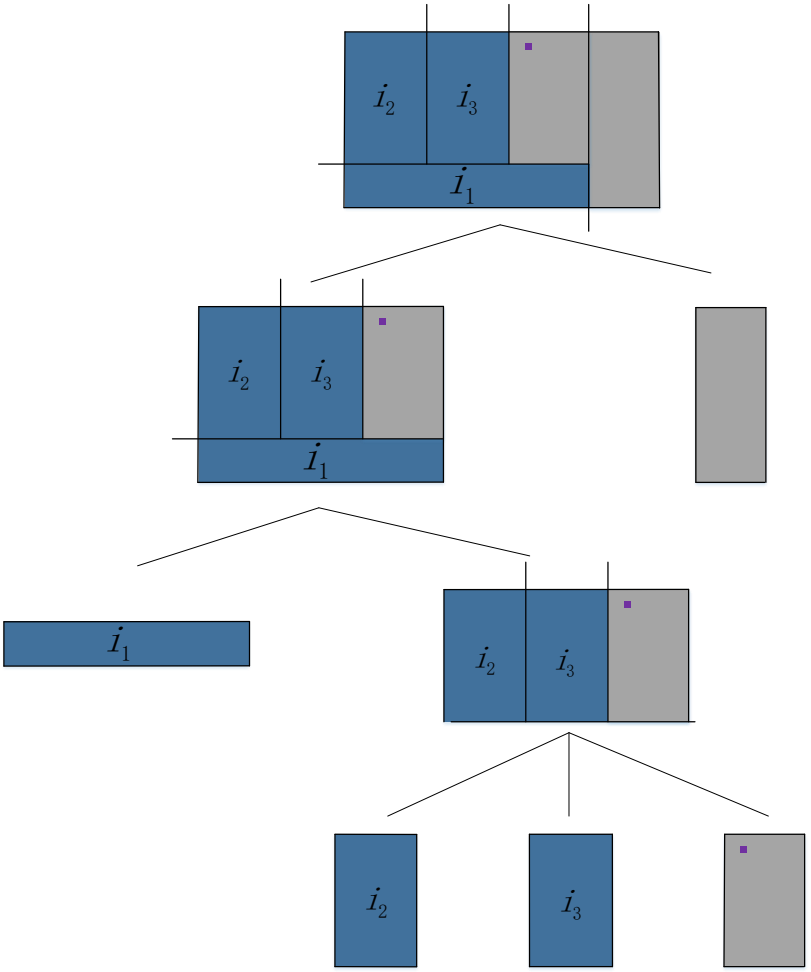


图 2-5 深度优先切割树

2.2 问题建模

具体来说，二维矩形切割优化问题的结构可由一个图表示。如图 2-6 所示，整个大矩形代表原料，其中蓝色矩形代表成品，浅灰色矩形代表废料，紫色方点代表瑕疵，黑色线段线就是切割轨迹，同时每一件成品的编号、尺寸、所属成品列表集和顺序给定；原料默认是有序的，只有前一块原料使用完毕，才能使用后续的原料，且原

料都是统一规格的，每一块原料上的每件瑕疵的坐标和尺寸已知。将一块原料切割完成后的情况就是上图中通过切割轨迹分割开来的蓝色和灰色矩形块。

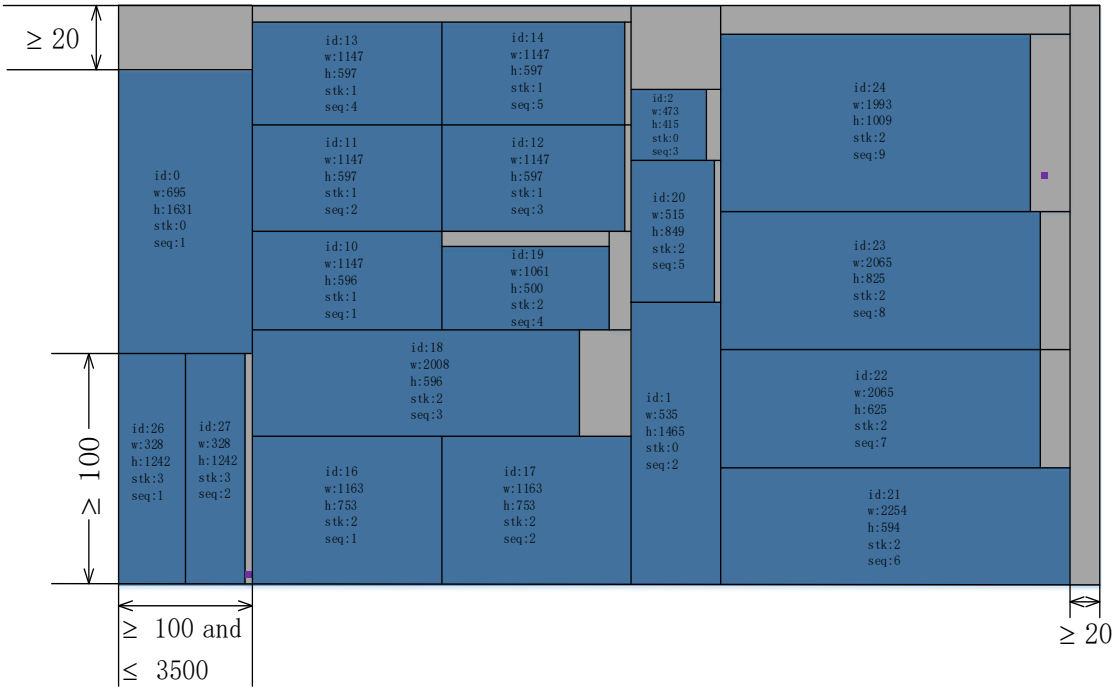


图 2-6 问题示例图

二维矩形切割优化问题的决策为切割得到这些成品需要多少原料，及每块原料上使用何种切割模式进行切割。该问题的目标是最小化原料的总用量。为了简化余料管理，通常我们只考虑最后一块原料上的余料情况。因此在计算总的原料用量时我们需要去掉最后一块原料上余料的尺寸。该问题的约束可以分为以下三个方面：

- **成品相关的约束：**
  - 成品可以旋转 90 度，即交换宽度和高度的值；
  - 必须切割完成订单列表集合中的每件订单，且每个订单恰好得到一件成品；
  - 成品不能过量生产，要满足指定的数量和尺寸；
  - 各有序订单列表中的成品完工顺序必须符合要求的顺序；
  - 订单列表中的某件成品订单不可更改到其他订单列表中；
- **原料相关的约束：**
  - 原料不允许旋转（旋转无意义，切割方案等价）；

- 各有序原料列表中的原料开工顺序必须符合指定的顺序；

- **切割模式相关的约束**
  - 任意一件成品必须完全包含在原料内；
  - 任意一对成品不能重叠；
  - 任意一件成品不能与任意一块瑕疵重叠；
  - 任意一刀的切割轨迹不能穿过任意一块瑕疵；
  - 任意成品最多经过三次切割得到其半成品， 再经过不超过一次的切割得到。第四刀可能恰好切割得到两件成品，或一件成品和一件废料；
  - 第一刀总是与高度方向平行；
  - 每一刀与前一刀的方向垂直；
  - 第一刀切割得到的包含成品的矩形区域宽度必须在给定区间范围内；
  - 第一刀切割得到的非余料的矩形区域宽度必须在给定区间范围内；
  - 第二刀切割得到的包含成品的矩形区域高度必须在给定区间范围内；
  - 任意一刀切割得到的不包含成品的矩形区域 （废料） 宽度和高度必须在给定区间范围内；

由于二维切割优化问题涉及到组合问题和生产调度问题，需要考虑的约束众多，由此通过深入分析研究，给出求解此问题的两种数学模型表示：完整模型（Complete Model，CM）、迭代模型（Iterative Model，IM）。本文提出两种数学规划模型的主要原因是：不同的模型具有不同的求解特性，根据各模型在不同算例上求解效果，可以比较分析数学模型的特性，加深对于建模方法的理解；另一方面模型的求解结果也可以用于分析和对比本文中提出的启发式算法的效率和性能。

2.2.1 完整模型

完整模型是一种最直观的建模方式，顾名思义它是完整地将问题中的涉及到所有已知、目标、约束和决策变量在建立模型之初就加入到模型中，后续在模型的求解过程中不会再继续添加。它实现起来往往比较简单，在实现模型算法之初我们会首先建立完整模型，便于加深对问题的理解，其他模型算法也大多是基于完整模型改进和

优化出来的衍生版本。另一方面，由于它完整地保留了问题中的相应约束和目标，当求解小较规模的算例时，它往往能得到比其他模型算法更好的结果。

表 2-1 集合的含义及描述

集合	描述	元素
I	成品集合	i, i'
S	有序订单列表集合	s
F	瑕疵集合	f
G	有序原料集合	g, g'
L^1	第一层（L1）虚拟桶集合	l, l'
L_l^2	第二层（L2）虚拟桶集合	m, m'
L_{lm}^3	第三层（L3）虚拟桶集合	n, n'

表 2-2 常量符号含义

常量	描述	标记
W	原料的长度	
H	原料的高度	
W_1^+	L1 虚拟桶的每一个有效解的最大宽度	
W_1^-	L2 虚拟桶的每一个有效解的最小宽度	
H_2^-	L2 虚拟桶的每一个有效解的最小高度	
W_3^-	L3 虚拟桶的每一个有效解的最小宽度	
H_4^-	L4 虚拟桶的每一个有效解的最小高度	
Ω_i, Ω_f	成品 i 或瑕疵 f 的宽度	$\Omega_i \neq 0$
E_i, E_f	成品 i 或瑕疵 f 的高度	$E_i \neq 0$
X_f	瑕疵 f 左下角的横坐标	在笛卡尔坐标系中
Y_f	瑕疵 f 左下角的纵坐标	在笛卡尔坐标系中
L	第三层虚拟桶的总数	$L = \|G\| \cdot \|L^1\| \cdot \|L_l^2\| \cdot \|L_{lm}^3\|$

表 2-3 决策变量

变量	描述	取值范围
d_i	成品 i 是否旋转了 90 度	{0,1}
p	原料上是否放置了成品	{0,1}
p_l	L1 虚拟桶 l 是否放置了成品	{0,1}
p_{lm}	L2 虚拟桶 (l,m)是否放置了成品	{0,1}
p_{lmn}	L3 虚拟桶 (l,m,n) 是否放置了成品	{0,1}

变量	描述	取值范围
p_{lmni}	成品 i 是否放置在 L3 虚拟桶 (l,m,n)	{0,1}
ω_l^1	L1 虚拟桶 l 的宽度	$[0,+\infty)$
η_{lm}^2	L2 虚拟桶 m 在 L1 虚拟桶 l 内的高度	$[0,+\infty)$
ω_{lmn}^3	L3 虚拟桶 n 在 L2 虚拟桶 (l,m) 内的宽度	$[0,+\infty)$
η_{lmn}^{4+}	L3 虚拟桶 (l,m,n) 中上部废料的高度	$[0,+\infty)$
η_{lmn}^{4-}	L3 虚拟桶 (l,m,n) 中下部废料的高度	$[0,+\infty)$
τ_{lm}^2	L2 虚拟桶 (l,m) 的高度是否是有效的	{0,1}
τ_{lmn}^3	L3 虚拟桶 (l,m,n) 的宽度是否是有效的	{0,1}
τ_{lmn}^{4+}	L3 虚拟桶 (l,m,n) 中上部废料是否是有效的	{0,1}
τ_{lmn}^{4-}	L3 虚拟桶 (l,m,n) 中下部废料是否是有效的	{0,1}
γ	最后一块已使用原料中使用区域的宽度	$[0,+\infty)$
e	原料中是否有余料	{0,1}
c_{lmnf}	L3 虚拟桶(l,m,n) 是否包含瑕疵 f	{0,1}
c_{lmnf}^k	L3 虚拟桶(l,m,n) 是否不存在 k^{th} 侧的瑕疵	{0,1}
c_f^1	原料中的余料是否包含瑕疵 f	{0,1}
o_i	成品 i 的生成顺序	$[0,+\infty)$

当待求解的问题规模较小时，可以建立完整的数学模型结合求解器直接求解。如上表 2-1 给出了问题中涉及的元素集合，表 2-2 给出了问题中已知常量的符号含义，表 2-3 给出了问题中决策变量的定义。该问题的完整数学模型如公式(2-1)至(2-46)所示。

- **目标：**
  - 最小化使用原料的宽度(objective of glass width，OGW)

问题的原始目标。只有原料在最后一次切割模式中执行最后一次切割后右侧剩余的废料才能重复使用（视为余料而不是废料）：

$$\min W \cdot \sum_{g \epsilon G} p_g - (W - \gamma) \tag{2-1}$$

- 最大化已放置成品的面积(objective of placed item，OPI)

该目标下的最优解是原始问题的可行解

$$\max \sum_{i \epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot \mathrm{p}(i) \tag{2-2}$$

- 最小化废料(objective of wasted glass，OWG)

如果每个成品必须放置，那么该目标等同于 OGW 目标

$$\min W\cdot\sum_{g\epsilon G}p_g-(W-\gamma)-\sum_{i\epsilon I}\Omega_i\cdot \mathrm{H}_i\cdot \mathrm{p}(i) \tag{2-3}$$

- **约束：**
  - 函数定义

在介绍模型中的相关约束前，我们先对模型中可能涉及到的相应函数进行说明

- $\mathrm{w}(i)$ 考虑了物品是否旋转后的实际宽度

$$\mathrm{w}(i)=\Omega_i\cdot(1-d_i)+H_i\cdot d_i \tag{2-4}$$

- $h(i)$ 考虑了物品是否旋转后的实际高度

$$h(i)=H_i\cdot(1-d_i)+\Omega_i\cdot d_i \tag{2-5}$$

- 函数 $\mathrm{x}(l,m,n)$ 表示 L3 虚拟桶左下侧的水平坐标

$$\mathrm{x}(l,m,n)=\sum_{l\prime\in L^1,l\prime<l}\omega^1_{l\prime}+\sum_{n\prime\in L^3_{lm},n\prime<n}\omega^3_{lmn\prime} \tag{2-6}$$

- 函数 $\mathrm{y}(l,m,n)$ 表示 L3 虚拟桶左下侧的垂直坐标

$$\mathrm{y}(l,m,n)=\sum_{m\prime\in L^2_l,m\prime<m}\eta^2_{lm\prime} \tag{2-7}$$

- 函数 $\mathrm{seq}(g,l,m,n)$ 表示原料块 g中 L3 虚拟桶 (l,m,n) 的生成顺序

$$\mathrm{seq}(g,l,m,n)=\left|L^3_{lm}\right|\cdot(|L^2_l|\cdot(|L^1|\cdot g+l)+m)+n \tag{2-8}$$

- 函数 $\mathrm{next}(i)$ 表示的是有序列表中的下一件成品 i
- 函数 $\mathrm{next}(g)$ 表示列表 G 中的下一块原料 g
- 函数 $\mathrm{p}(i)$ 表示成品 i 是否放置在原料上

$$\mathrm{p}(i)=\sum_{l\in L^1}\sum_{m\in L^2_l}\sum_{n\in L^3_{lm}}p_{lmni} \tag{2-9}$$

通过前文中的分析，二维矩形切割问题的优化和二维装箱问题有相似之处，所以在建立数学模型时，考虑了这一点，将切割模式的相关约束转换为装箱问题对应的约束来处理，它们在实现效果上等同。接下来将从九个方面来介绍该问题的约束。

- 辅助约束
  - 原料的使用约束：是否有成品放置在原料内。如果禁用 OPI 目标，则可以忽略不等式左侧约束

$$p \le \sum_{i\in I}\sum_{l\in L^1}\sum_{m\in L_l^2}\sum_{n\in L_{lm}^3} p_{lmni} \le |I|\cdot p \tag{2-10}$$

  - L1 虚拟桶放置约束：是否有成品放置在 L1 虚拟桶 l 内。如果禁用 OPI 目标，则可以忽略不等式左侧约束

$$p_l \le \sum_{i\in I}\sum_{m\in L_l^2}\sum_{n\in L_{lm}^3} p_{lmni} \le |I|\cdot p_l, \quad \forall l \in L^1 \tag{2-11}$$

  - L2 虚拟桶放置约束：是否有成品放置在 L2 虚拟桶 (l,m) 内。如果禁用 OPI 目标，则可以忽略不等式左侧约束

$$p_{lm} \le \sum_{i\in I}\sum_{n\in L_{lm}^3} p_{lmni} \le |I|\cdot p_{lm}, \quad \forall l \in L^1, \forall m \in L_l^2 \tag{2-12}$$

  - L3 虚拟桶放置约束：是否有成品放置在 L3 虚拟桶 (l,m,n) 内。如果禁用 OPI 目标，则可以忽略左侧约束；如果加入唯一性放置约束，不等式右 $|I|$ 侧简化为 1

$$p_{lmn} \le \sum_{i\in I} p_{lmni} \le |I|\cdot p_{lmn}, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \tag{2-13}$$

  - 唯一性放置约束：在单个的 L3 虚拟桶 (l,m,n) 里放置的成品不能超过 1，如果启用无瑕疵约束，则该约束是无效的

$$\sum_{i\in I} p_{lmni} \le 1, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \tag{2-14}$$

- 组合约束
  - 余料位置约束：将余料定位在最后一块原料上并获得所用区域的宽度，如果 g 是最后一块原料，那么将 g' 设定为 0，因为最后一块原料的下一块原料永远不会被使用

$$\sum_{l\in L_g^1} w_{gl} - W\cdot\left(1-p_g+p_g'\right) \le \gamma \le \sum_{l\in L_g^1} w_{gl} + W\cdot\left(1-p_g+p_g'\right), \tag{2-15}$$

$$\forall g, g' \in G, g' = \text{next}(g)$$

- L1 虚拟桶的总宽度约束：所有 L1 虚拟桶的宽度之和应该等于原料的宽度或为最后一块原料的使用宽度。只有最后一块原料上最右边的废料是余料，因此需要为剩余的原料考虑 W_1^+ 限制。

$$\text{W} \cdot (1 - \text{e}) \leq \sum_{l \epsilon L} \omega_l^1 \leq W - W_3^- \cdot e \tag{2-16}$$

- L2 虚拟桶的总高度约束：所有 L2 虚拟桶的高度之和应该等于原料的高度

$$\sum_{m \epsilon L_l^2} \eta_{lm}^2 = H, \quad \forall l \in L^1 \tag{2-17}$$

- L3 虚拟桶的总宽度约束：所有 L3 虚拟桶的宽度之和不应超过其所在的 L1 虚拟桶的宽度。

$$\sum_{n \epsilon L_l^3} \omega_{lmn}^3 = \omega_l^1, \quad \forall l \in L^1, \forall m \in L^2 \tag{2-18}$$

■ 拟合约束

- 水平拟合约束：如果成品 i 已被放置，成品 i 的宽度应该等于 L3 虚拟桶 (l, m, n) 的宽度。不等式左侧的 W 可以减小到 Ω_i 来收紧边界，但它可能不会加速优化；不等式右侧的 W 可以是不小于 ω_{lmn}^3 的任何值

$$\begin{aligned} &\text{w}(i) - W \cdot (1 - p_{lmni}) \leq \omega_{lmn}^3 \leq \text{w}(i) + W \cdot (1 - p_{lmni}), \\ &\forall i \in I, \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \end{aligned} \tag{2-19}$$

- 垂直拟合约束：如果成品 i 已被放置，成品 i 的高度应该等于 L3 虚拟桶 (l, m, n) 的高度。第一个不等式左侧的 H 可以减小到Ω_i来收紧边界，但它可能不会加速优化；第二个不等式右侧的 H 可以是不小于 η_{lmnk}^2 的任何值

$$\begin{aligned} &h(i) - H \cdot (1 - p_{lmni}) \leq \eta_{lm}^2 - \eta_{lmn}^4 - \eta_{lmn}^{4-} \\ &\eta_{lm}^2 - \eta_{lmn}^{4+} - \eta_{lmn}^{4-} \leq \eta_{lm}^2 \leq h(i) + H \cdot (1 - p_{lmni}), \\ &\forall i \in I, \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \end{aligned} \tag{2-20}$$

- 单一放置约束：每个成品都应该放在一个 L3 虚拟桶中。如果启用 OPI 目标，则等号应该变为不等号

$$\sum_{l\in L^1}\sum_{m\in L_l^2}\sum_{n\in L_{lm}^3} p_{lmni} = 1, \quad \forall i \in I \tag{2-21}$$

■ 边界约束

- L1 宽度边界约束：如果放置了成品，则 L1 虚拟桶的宽度不应超过宽度限制。不等式左侧可以由应用于每个 L2 虚拟桶、L3 虚拟桶或 L3 虚拟桶中的成品的一系列约束替换

$$W_1^- \cdot p_l \le \omega_l^1 \le W_1^+, \quad \forall l \in L^1 \tag{2-22}$$

- L2 最小高度约束：L2 虚拟桶 (l,m) 的高度不应小于最小高度。不等式左侧的 H 可以减小到 H_4^- 来收紧边界

$$\begin{aligned} H_2^- \cdot p_{lm} \le \eta_{lm}^2, \quad \forall l \in L^1, \forall m \in L_l^2 \\ H_4^- \cdot \tau_{lm}^2 - H \cdot p_{lm} \le \eta_{lm}^2 \le H \cdot \tau_{lm}^2 + H \cdot p_{lm}, \quad \forall l \in L^1, \forall m \in L_l^2 \end{aligned} \tag{2-23}$$

- L3 最小宽度约束：L3 虚拟桶 (l,m,n)如果为空，则其宽度不应小于最小宽度。不等式左侧可以由应用于 L3 虚拟桶中的每个成品的一系列约束来替换。如果所有成品的宽度和高度都不小于 W_3^-, H_4^-，则可以省略 $W \cdot p_{lmn}$

$$\begin{aligned} W_3^- \cdot \tau_{lmn}^3 - W \cdot p_{lmn} \le \omega_{lmn}^3 \le W \cdot \tau_{lmn}^3 + W \cdot p_{lmn}, \\ \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \end{aligned} \tag{2-24}$$

- L4 最小高度约束：L3 虚拟桶 (l,m,n) 中的废料高度不应小于最小高度

$$\begin{aligned} H_4^- \cdot \tau_{lmn}^{4+} \le \eta_{lmn}^{4+} \le H \cdot \tau_{lmn}^{4+}, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \\ H_4^- \cdot \tau_{lmn}^{4-} \le \eta_{lmn}^{4-} \le H \cdot \tau_{lmn}^{4-}, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \end{aligned} \tag{2-25}$$

- 最大瑕疵约束：L3 虚拟桶 (l,m,n) 中应该有不超过一块废料

$$\tau_{lmn}^{4+} + \tau_{lmn}^{4-} \le 1, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \tag{2-26}$$

■ 瑕疵约束

- 无瑕疵约束：如果 L3 虚拟桶 (l,m,n) 中包含瑕疵，则不应放置任何

成品。如果加入唯一性放置约束，该约束将无效

$$\sum_{i\in I} p_{lmni} \le 1 - c_{lmnf}, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall f \in F \tag{2-27}$$

- 包含瑕疵约束：如果 L3 虚拟桶 (l,m,n) 不在瑕疵 f 的任一侧，即它们是重叠的，则 L3 包含瑕疵 f。

$$c_{lmnf} \ge \bigwedge_{k=1}^{4} c_{lmnf}^k \;\Leftrightarrow\; 3 + c_{lmnf} \ge \sum_{k=1}^{4} c_{lmnf}^k, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall f \in F \tag{2-28}$$

- 瑕疵方位约束：如果 L4 虚拟桶 (l,m,n) 不在瑕疵 f 的某些方位，则 c_{lmnf}^k 值为真。
  - 对于 L4 虚拟桶

$$\mathrm{x}(l,m,n) + W \cdot c_{lmnf}^1 \ge X_f + \Omega_f, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall f \in F \tag{2-29}$$

$$\mathrm{x}(l,m,n) + \omega_{lmn}^3 - W \cdot c_{lmnf}^2 \le X_f, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall f \in F \tag{2-30}$$

$$\mathrm{y}(l,m,n) + \eta_{lmn}^{4-} + H \cdot c_{lmnf}^3 \ge Y_f + \mathrm{H}_f, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall f \in F \tag{2-31}$$

$$\mathrm{y}(l,m,n) + \eta_{lm}^2 - \eta_{lmn}^{4+} - H \cdot c_{lmnf}^4 \le Y_f, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall f \in F \tag{2-32}$$

■ 次序约束

- 成品次序约束：如果成品 i 放置在 L3 虚拟桶 (l,m,n) 中，则其有序列表中的下一个成品只能在 L3 虚拟桶 (l',m',n') 中，其中 $\mathrm{seq}(l,m,n) < \mathrm{seq}(l',m',n')$。不等式右边可以是 $\sum p_{l'm'n'i'}$，其中 $(l',m',n') \in \{(l',m',n') | \mathrm{seq}(l,m,n) < \mathrm{seq}(l',m',n')\}$来减少约束的数量

$$1 - p_{lmni} \ge p_{l'm'n'i'}, \quad \forall l,l' \in L^1, \forall m,m' \in L_l^2, \forall n,n' \in L_{lm}^3, \forall i,i' \in I, i' = \mathrm{next}(i), \quad (l' \le l-1) \vee \Big((l'=l) \wedge \big((m' \le m-1) \vee ((m'=m) \wedge (n' \le n-1))\big)\Big) \tag{2-33}$$

- 放置约束：如果未放置前前序成品，则无法放置当前成品。如果禁用

OPI 目标，则该约束可以忽略。

$$\sum_{l\in L^1}\sum_{m\in L_l^2}\sum_{n\in L_{lm}^3} p_{lmni} \geq \sum_{l\in L^1}\sum_{m\in L_l^2}\sum_{n\in L_{lm}^3} p_{l'm'n'i'}, \forall i, i' \in I, i' = \text{next}(i) \tag{2-34}$$

- 虚拟桶次序约束：
  - L1 虚拟桶水平次序约束：将所有较小的 L1 虚拟桶放在最右边

$$\tau_{lmn}^3 \geq \tau_{l'mn}^3, \quad \forall l, l' \in L^1, l' = l+1, m = 0, n = 0 \tag{2-35}$$

如果没有瑕疵和顺序要求，设置 $W = 1$ 使得宽度值更大的虚拟桶首先置。

$$W \cdot \omega_l^1 \geq \omega_{l'}^1, \quad \forall l, l' \in L^1, l' = l+1 \tag{2-36}$$

  - L2 虚拟桶垂直次序约束：将所有较小的 L2 虚拟桶放在最上边

$$\tau_{lm}^2 \geq \tau_{lm'}^2, \quad \forall l \in L^1, \forall m, m' \in L_l^2, m' = m+1 \tag{2-37}$$

如果没有瑕疵和顺序要求，设置 $H = 1$ 使得高度值更大的虚拟桶首先放置。

$$H \cdot \eta_{lm}^2 \geq \eta_{lm'}^2, \quad \forall l \in L^1, \forall m, m' \in L_l^2, m' = m+1 \tag{2-38}$$

  - L3 虚拟桶水平次序约束：将所有较小的 L3 虚拟桶放在最右边

$$\tau_{lmn}^3 \geq \eta_{lmn'}^3, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n, n' \in L_{lm}^3, n' = n+1 \tag{2-39}$$

$$W \cdot \omega_{lmn}^3 \geq \omega_{lmn'}^3, \quad \forall l \in L^1, \forall m, m' \in L_l^2, \forall n, n' \in L_{lm}^3, n' = n+1 \tag{2-40}$$

- 空虚拟桶合并约束：
  - L1 虚拟桶合并约束：不应该有连续的较小的空 L1 虚拟桶。该约束可能会剪掉最优解，当存在一些瑕疵使得区域宽度大于 W_1^+ 则该区域不能放置任何成品；如果启用 L1 虚拟桶水平次序约束，该约束可能会剪掉更多对称解。

$$p_l + p_{l'} \geq \tau_{l'mn}^3, \quad \forall l, l' \in L^1, l' = l+1, m = 0, n = 0 \tag{2-41}$$

  - L2 虚拟桶合并约束：不应该有连续的较小的空 L2 虚拟桶。如果启用

L2 虚拟桶垂直次序约束，它可能会剪掉更多对称解。

$$p_{lm} + p_{lm'} \geq \tau^2_{lm'}, \quad \forall l \in L^1, \forall m, m' \in L^2_l, m' = m + 1 \tag{2-42}$$

- L3 虚拟桶合并约束：不应该有连续的较小的空 L3 虚拟桶。如果启用 L3 虚拟桶水平次序约束，它可能会剪掉更多对称解。

$$p_{lmn} + p_{lmn'} \geq \tau^3_{lmn'}, \quad \forall l \in L^1, \forall m \in L^2_l, \forall n, n' \in L^3_{lm}, n' = n + 1 \tag{2-43}$$

- 原料约束
  - 原料顺序约束：原料应该一块接一块地使用，而不能跳过一些原料块

$$p_g \geq p_{g'}, \quad \forall g, g' \in G, g' = \text{next}(g) \tag{2-44}$$

  - 区域边界约束：成品覆盖面积应小于原料面积

$$\sum_{i \epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot \mathrm{p}(i) \leq W \cdot H \tag{2-45}$$

  - 最小宽度约束：使用原料面积的总宽度乘以原料高度应不小于成品总覆盖面积。不等式左侧可以用 $H \cdot \sum_{l \in L^1} \omega^1_l$ 替代来收紧边界，但无法提高性能

$$H \cdot \left(W \cdot \sum_{g \in G} p_g - (W - \gamma) \right) \geq \sum_{i \epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot \mathrm{p}(i) \tag{2-46}$$

2.2.2 迭代模型

当求解问题规模较大时，完整模型的规模变得异常庞大，模型预处理需要消耗大量的时间，使得模型的求解效率急剧下降。针对这种情况的算例我们提出了一种迭代思想的模型：将问题拆分成子问题，再针对子问题调用模型迭代求解。子问题的划分主要分为两块：首先，每次填充 k 块原料 (k 可能小于 1)，最大化放置成品总面积；然后，每次将所有有序订单列表的第一件未放置成品放入，并最大化利用率 (放置成品总面积/使用原料宽度)。这样在每一轮求解过程中问题的规模大大减小，每轮中模型需要加载的数据也大大降低，因此模型的求解效率得到了提升，即使需要经过多轮迭代，但是它的求解效率依然明显优于完整模型。如下表 2-4 给出了问题中涉及的元素集合，表 2-5 给出了问题中已知常量的符号含义，表 2-6 给出了问题中决

策变量的定义。该问题的迭代数学模型如下所示。

- **已知：**满足上表 2-1、表 2-2 所示的已知条件外，新增如下内容：

表 2-4 新增集合的含义及描述

集合	描述	元素
L^4_{lmn}	第四层（L4）虚拟桶集合	k, k'

表 2-5 新增常量符号含义

常量	描述	标记
U	估计的最佳利用率	参数

- **决策变量：**满足上表 2-3 所示的决策变量外，删除了 p_{lmni}、η^{4+}_{lmn}、η^{4-}_{lmn}、τ^{2}_{lm}、τ^{4+}_{lmn}、τ^{4-}_{lmn}、c_{lmnf}、c^{k}_{lmnf} 新增如下内容：

表 2-6 新增决策变量

变量	描述	取值范围
p_{lmnk}	L4 虚拟桶 (l,m,n,k) 是否放置了成品	$\{0,1\}$
p_{lmnki}	成品 i 是否放置在 L4 虚拟桶 (l,m,n,k)	$\{0,1\}$
η^4_{lmnk}	L4 虚拟桶 k 在 L3 虚拟桶 (l,m,n) 的高度	$[0,+\infty)$
τ^4_{lmnk}	L4 虚拟桶 (l,m,n,k) 的高度是否是有效的	$\{0,1\}$
c_{lmnkf}	L4 虚拟桶(l,m,n,k) 是否包含瑕疵 f	$\{0,1\}$

- 满足已有目标公式(2-1)(2-2)(2-3)，新增如下目标
  - 最大化利用率（objective of utilization ratio，OUR）
    - 原始表示：如果每个成品必须放置，那么该目标等同于 OGW 目标

$$\max \frac{\sum_{i\epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot \mathrm{p}(i)}{\min W \cdot \sum_{g\in G} p_g - (W-\gamma)} \tag{2-47}$$

    - 线性表示：通过引入最佳利用率 U 的估计，可以将公式(2-56)转换为线性形式。如果 U 被高估，则可能需要增加额外约束以强制放置至少 1 个成品。用 U 值代替当前目标值，并重新优化直到 U 收敛，可以得到更好的结果。

$$\max \sum_{i \epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot \mathrm{p}(i) - U \cdot W \cdot \sum_{g \in G} p_g - (W - \gamma) \tag{2-48}$$

◆ 最小化使用原料的估计宽度（objective of estimated width，OEW）

$$\min W \cdot \sum_{g \in G} p_g - (W - \gamma) + U^{-1} \cdot \sum_{i \epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot (1 - p(i)) \tag{2-49}$$

- **约束**

■ 函数定义

我们对模型中可能涉及到的相应函数进行说明：由完整模型函数(2-4)(2-5)(2-6)和$\mathrm{next}(i)$、$\mathrm{next}(g)$和如下函数组成

◆ 函数 $\mathrm{y}(l,m,n)$ 表示 L3 虚拟桶左下侧的垂直坐标

$$\mathrm{y}(l,m,n) = \sum_{m\prime \in L_l^2, m\prime < m} \eta_{lm\prime}^2 + \sum_{k\prime \in L_{lmn}^4, k\prime < k} \eta_{lmnk\prime}^4 \tag{2-50}$$

◆ 函数 $\mathrm{seq}(g,l,m,n)$ 表示原料块 g中 L3 虚拟桶 (l,m,n) 的生成顺序

$$\mathrm{seq}(g,l,m,n) = (((g \cdot |L^1| + l) \cdot |L_l^2| + m) \cdot |L_{lm}^3| + n) \cdot |L_{lmn}^4| + k \tag{2-51}$$

◆ 函数 $\mathrm{p}(i)$ 表示成品 i 是否放置在原料上

$$\mathrm{p}(i) = \sum_{l \in L^1} \sum_{m \in L_l^2} \sum_{n \in L_{lm}^3} \sum_{n \in L_{lmn}^4} p_{lmnki} \tag{2-52}$$

■ 辅助约束：

◆ 原料中成品放置约束：是否有成品放置在原料内。如果禁用 OPI 目标，则可以忽略左侧约束

$$p \le \sum_{l \in L^1} p_l \le |I| \cdot p \tag{2-53}$$

◆ L1 虚拟桶放置约束：是否有成品放置在 L1 虚拟桶内。如果禁用 OPI 目标，则可以忽略左侧约束

$$p_l \le \sum_{m \in L_l^2} p_{lm} \le |I| \cdot p_l, \qquad \forall l \in L^1 \tag{2-54}$$

◆ L2 虚拟桶放置约束：是否有成品放置在 L2 虚拟桶内。如果禁用 OPI 目标，则可以忽略左侧约束

$$p_{lm} \le \sum_{n \in L_{lm}^3} p_{lmn} \le |I| \cdot p_{lm}, \qquad \forall l \in L^1, \forall m \in L_l^2 \tag{2-55}$$

◆ L3 虚拟桶放置约束：是否有成品放置在 L3 虚拟桶内。如果禁用 OPI 目标，则可以忽略左侧约束

$$p_{lmn} \le \sum_{k \in L_{lmn}^4} p_{lmnk} \le |I| \cdot p_{lmn}, \qquad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \tag{2-56}$$

◆ L1 虚拟桶放置约束：是否有成品放置在 L4 虚拟桶内。如果禁用 OPI 目标，则可以忽略左侧约束；如果加入唯一性放置约束，右侧简化为 1

$$p_{lmnk} \le \sum_{i \in I} p_{lmnki} \le |I| \cdot p_{lmnk}, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall k \in L_{lmn}^4 \tag{2-57}$$

◆ 唯一性放置约束：在单个的 L4 虚拟桶 (l, m, n, k) 里放置的成品不能超过 1，如果没有瑕疵，则该约束是无效的

$$\sum_{i \in I} p_{lmnki} \le 1, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall k \in L_{lmn}^4 \tag{2-58}$$

■ 组合约束：由完整模型约束(2-16)、(2-17)、(2-18)和如下约束组成

◆ L4 虚拟桶的总高度约束：所有 L4 虚拟桶的高度之总和应该等于其所在 L2 虚拟桶的高度。

$$\sum_{k \in L_{lmn}^4} \eta_{lmnk}^4 = \eta_{lm}^2, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3 \tag{2-59}$$

■ 拟合约束

◆ 水平拟合约束：如果成品 i 已被放置，成品 i 的宽度应该等于 L4 虚拟桶 (l, m, n, k) 的宽度。不等式左侧的 W 可以减小到 Ω_i 来收紧边界，但它可能不会加速优化；不等式右侧的 W 可以是不小于 ω_{lmn}^3 的任何值。

$$w(i) - W \cdot (1 - p_{lmnki}) \le \omega_{lmn}^3 \le w(i) + W \cdot (1 - p_{lmnki}), \quad \forall i \in I, \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall k \in L_{lmn}^4 \tag{2-60}$$

◆ 垂直拟合约束：如果成品 i 已被放置，成品 i 的高度应该等于 L4 虚

拟桶 (l,m,n,k) 的高度。不等式左侧的 H 可以减小到 Ω_i 来收紧边界，但它可能不会加速优化；不等式右侧的 H 可以是不小于 η^4_{lmnk} 的任何值。

$$h(i)-H\cdot(1-p_{lmnki})\le\eta^4_{lmnk}\le h(i)+H\cdot(1-p_{lmnki}),\quad \forall i\in I,\forall l\in L^1,\forall m\in L^2_l,\forall n\in L^3_{lm},\forall k\in L^4_{lmn} \tag{2-61}$$

- 单一放置约束：每个成品都应该放在一个 L4 虚拟桶中。如果启用 OPI 目标，则等号应该变为不等号。

$$\sum_{l\in L^1}\sum_{m\in L^2_l}\sum_{n\in L^3_{lm}}\sum_{n\in L^4_{lmn}}p_{lmnki}=1,\quad \forall i\in I \tag{2-62}$$

- 边界约束：由完整模型约束(2-22)、(2-24)和如下约束组成
  - L2 最小高度约束：如果放置了成品，L2 虚拟桶 (l,m) 的高度不应小于最小高度

$$H_2^-\cdot p_{lm}\le\eta^2_{lm},\quad \forall l\in L^1,\forall m\in \tag{2-63}$$

  - L4 最小高度约束：如果 L4 虚拟桶 (l,m,n,k) 为空，则它的高度不应小于最小高度。如果所有成品的高度不小于 H_4^-，则可以省略不等式中的 $H\cdot p_{lmnk}$。不等式左侧的 H 可以减少到 H_4^- 来收紧边界。

$$H_4^-\cdot\tau^4_{lmnk}-H\cdot p_{lmnk}\le H\cdot\tau^4_{lmnk}+H\cdot p_{lmnk},\quad \forall l\in L^1,\forall m\in L^2_l,\forall n\in L^3_{lm},\forall k\in L^4_{lmn} \tag{2-64}$$

  - 无效的继承约束：如果 L4 虚拟桶 (l,m,n,k) 的父 L3 虚拟桶(l,m,n)是无效的，那么它也是无效的。

$$\tau^4_{lmnk}\le\tau^3_{lmn},\quad \forall l\in L^1,\forall m\in L^2_l,\forall n\in L^3_{lm},\forall k\in L^4_{lmn} \tag{2-65}$$

- 瑕疵约束：
  - 无瑕疵约束：如果 L4 虚拟桶 (l,m,n,k) 中包含瑕疵，则不应放置任何成品。如果加入唯一性放置约束，该约束将无效。

$$\sum_{i\in I}p_{lmnki}\le 1-c_{lmnkf},\quad \forall l\in L^1,\forall m\in L^2_l,\forall n\in L^3_{lm},\forall k\in L^4_{lmn},\forall f\in F \tag{2-66}$$

  - 瑕疵旁路约束：禁止切割轨迹经过瑕疵，即每个瑕疵应由单个 L4 虚拟

桶或余料部分完全覆盖。

$$c_f^1 + \sum_{l\in L^1}\sum_{m\in L_l^2}\sum_{n\in L_{lm}^3}\sum_{k\in L_{lmn}^4} c_{lmnkf} = 1, \quad \forall f \in F \tag{2-67}$$

◆ 瑕疵覆盖约束：如果 L4 虚拟桶 (l,m,n,k) 不覆盖瑕疵 f，则 c_{lmnfk} 的值为假。这种约束不能保证其逆否命题成立。

- 对于 L4 虚拟桶：

$$\begin{gathered}
\mathrm{x}(l,m,n) - W\cdot\left(1-c_{lmnkf}\right) \le X_f, \\
\forall l\in L^1, \forall m\in L_l^2, \forall n\in L_{lm}^3, \forall k\in L_{lmn}^4, \forall f\in F \\
\mathrm{x}(l,m,n) + \omega_{lmn}^3 + W\cdot\left(1-c_{lmnkf}\right) \ge X_f + \Omega_f, \\
\forall l\in L^1, \forall m\in L_l^2, \forall n\in L_{lm}^3, \forall n\in L_{lmn}^4, \forall f\in F \\
\mathrm{y}(l,m,n,k) - H\cdot\left(1-c_{lmnkf}\right) \le Y_f, \\
\forall l\in L^1, \forall m\in L_l^2, \forall n\in L_{lm}^3, \forall n\in L_{lmn}^4, \forall f\in F \\
\mathrm{y}(l,m,n,k) + H\cdot\left(1-c_{lmnkf}\right) \ge Y_f + \mathrm{H}_f, \\
\forall l\in L^1, \forall m\in L_l^2, \forall n\in L_{lm}^3, \forall n\in L_{lmn}^4, \forall f\in F
\end{gathered} \tag{2-68}$$

- 对于剩余的原料：

$$\sum_{l\in L^1}\omega_l^1 - W\cdot\left(1-c_f^1\right) \le X_f, \quad \forall f\in F \tag{2-69}$$

◆ 瑕疵拟合约束：瑕疵只能由有效的 L4 虚拟桶覆盖。

$$c_{lmnkf} \le \tau_{lmnk}^4, \quad \forall l\in L^1, \forall m\in L_l^2, \forall n\in L_{lm}^3, \forall k\in L_{lmn}^4, \forall f\in F \tag{2-70}$$

■ 次序约束：

◆ 成品次序约束：如果成品 i 被放置在 L4 虚拟桶 (l,m,n,k) 中，则其有序列表中的下一个成品只能在 L4 虚拟桶 (l',m',n',k') 中，其中 $\mathrm{seq}(l,m,n,k) < \mathrm{seq}(l',m',n',k')$。不等式右边可以是 $\sum p_{l'm'n'k'i'}$，其中 $(l',m',n',k') \in \{(l',m',n',k')|\mathrm{seq}(l,m,n,k) < \mathrm{seq}(l',m',n',k')\}$来减少约束的数量。

$$\begin{gathered}
1 - p_{lmnki} \ge p_{l'm'n'k'i'}, \\
\forall l,l'\in L^1, \forall m,m'\in L_l^2, \forall n,n'\in L_{lm}^3, k'\in L_{lmn}^4, \forall i,i'\in I,
\end{gathered} \tag{2-71}$$

$$i' = \text{next}(i), (l' \le l-1) \vee ((l' = l) \wedge \left(\begin{array}{c} (m' \le m-1) \\ \vee \left((m' = m) \wedge \left((n' \le n-1) \vee \left((n = n') \wedge (k \le k') \right) \right) \right) \end{array} \right))$$

◆ 放置约束：如果未放置其前序成品，则无法放置当前成品。如果禁用 OPI 目标，则该约束可以省略。

$$\sum_{l \in L^1} \sum_{m \in L_l^2} \sum_{n \in L_{lm}^3} \sum_{k \in L_{lmn}^4} p_{lmnki} \ge \sum_{l \in L^1} \sum_{m \in L_l^2} \sum_{n \in L_{lm}^3} \sum_{k \in L_{lmn}^4} p_{lmnki'}, \quad \forall i, i' \in I, i' = \text{next}(i) \tag{2-72}$$

■ 虚拟桶次序约束：由完整模型约束(2-35)、(2-36)、(2-39)、(2-40)和如下约束组成

◆ L2 虚拟桶垂直次序约束：将所有较小的 L2 虚拟桶放在最上侧

$$\tau_{lmnk}^4 \ge \tau_{lm'nk}^4, \quad \forall l \in L^1, \forall m, m' \in L_l^2, m' = m+1, n = 0, k = 0 \tag{2-73}$$

如果没有瑕疵和顺序要求，设置 $H = 1$ 以使高度值更大的虚拟桶首先放置。

$$H \cdot \eta_{lm}^2 \ge \eta_{lm'}^2, \quad \forall l \in L^1, \forall m, m' \in L_l^2, m' = m+1 \tag{2-74}$$

◆ L4 虚拟桶垂直次序约束：将所有较小的 L4 虚拟桶放在最上侧

$$\tau_{lmnk}^4 \ge \tau_{lmnk'}^4, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall k, k' \in L_{lmn}^4, k' = k+1 \tag{2-75}$$

如果没有瑕疵和顺序要求，设置 $H = 1$ 以使高度值更大的虚拟桶首先放置。

$$H \cdot \eta_{lmnk}^4 \ge \eta_{lmnk'}^4, \quad \forall l \in L^1, \forall m \in L_l^2, \forall n \in L_{lm}^3, \forall k, k' \in L_{lmn}^4, k' = k+1 \tag{2-76}$$

■ 空虚拟桶合并约束：由完整模型约束(2-41)、(2-43)和如下约束组成

◆ L2 虚拟桶合并约束：不应该有连续的较小的空 L2 虚拟桶。如果启用 L2 虚拟桶垂直次序约束，它可能会去掉更多对称解。

$$p_{lm} + p_{lm'} \ge \tau_{lm'nk}^4, \quad \forall l \in L^1, \forall m, m' \in L_l^2, m' = m+1, n = 0, k = 0 \tag{2-77}$$

◆ L4 虚拟桶合并约束：不应该有连续的较小的空 L4 虚拟桶。如果启用 L4 虚拟桶垂直次序约束，它可能会去掉更多对称解。

$$p_{lmnk} + p_{lmnk'} \geq \tau^4_{lmnk'},$$
$$\forall l \in L^1, \forall m \in L^2_l, \forall n \in L^3_{lm}, \forall k, k' \in L^4_{lmn}, k' = k + 1 \tag{2-78}$$

- 原料的约束：由完整模型约束(2-44)、(2-46)和如下约束组成
  - 原料区域边界约束：成品覆盖面积应该小于原料面积

$$\sum_{i \epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot \sum_{l \in L^1} \sum_{m \in L^2_l} \sum_{n \in L^3_{lm}} \sum_{k \in L^4_{lmn}} p_{glmnki} \leq W \cdot H, \forall g \in G \tag{2-79}$$

  - L1 区域边界约束：成品覆盖面积应小于 L1 虚拟桶所在区域的面积

$$\sum_{i \epsilon I} \Omega_i \cdot \mathrm{H}_i \cdot \sum_{m \in L^2_l} \sum_{n \in L^3_{lm}} \sum_{k \in L^4_{lmn}} p_{lmnki} \leq W^+ \cdot H, \forall l \in L_1 \tag{2-80}$$

2.3 本章小结

本章先介绍了二维矩形切割优化模型的问题来源，简单描述了业务场景，接着介绍该问题中的相关概念和约定问题中的相应数据。再将问题进行抽象和转换，用图的语言形式化地描述整个问题，讲清楚图上的情况，再讲对应到具体业务场景的相关内容。最后对该问题建立了两种数学模型：完整模型和迭代模型来应对不同规模的算例。给出模型中已知、决策变量、目标和约束等数学语言的描述。

3 求解 TDRCO 问题的启发式树搜索算法

在本章将详细地介绍求解二维矩形切割优化问题的启发式树搜索算法。我们尝试使用通用的树搜索方法，但随着问题规模的增大，搜索整个解空间受限于性能和效率。因此，对于二维矩形切割优化问题，本文设计了一种迭代的带扰动机制和分支定界的启发式树搜索算法。分支定界结合深度优先算法是解决组合优化问题的常用算法，算法实现起来较为简单，但是如果能深入理解问题，制定合适的分支和剪枝策略也往往有比较好的结果。为了提高求解优度和求解效率，我们通过反复迭代对解空间进行优化，并且使用扰动策略来避免算法陷入次优解的死循环。下面首先介绍启发式树搜索算法的具体流程，然后再结合实际研究的二维矩形切割优化问题来详细描述针对该问题的求解算法。

3.1 启发式树搜索算法概述

树结构是链接节点的层次结构，其中每个节点表示特定状态。节点没有子节点，或有一个或多个子节点。解决方案是从“根”节点（表示初始状态）到“目标”节点（表示期望状态）的路径。树搜索算法试图通过遍历树结构来找到解决方案：从根节点开始以系统方式检查（扩展）子节点，如果保证找到解决方案（如果存在），则搜索策略完成。如果存在多个解决方案时保证找到最佳解决方案，则搜索策略是最佳的。

- 盲搜索算法：使用固定策略来有条理地遍历搜索树，如：广度优先（Breadth-first Search，BFS）[42]和深度优先搜索（Depth-first Search）[43]。盲搜索不适用于复杂问题，因为大的搜索空间（不同的搜索状态的数量）使得它们在有限的时间和空间中求解显得不切实际。
- 启发式搜索算法：使用启发式函数来确定遍历节点的顺序，优先考虑被判断为最有可能达到所需目标的状态，例如蒙特卡洛树搜索（Monte Carlo tree search，MCTS）[44]算法。使用启发式搜索策略可将搜索空间缩小到更易于管理的大小。

深度优先算法常用于解的构造，但对于求解大规模、复杂组合优化问题时，通用的树搜索算法在搜索解空间上显得无能为力，而分支定界法的分支和剪枝策略刚好是为了缩小解空间而设计的。因此我们考虑将通用的树搜索算法与分支定界法相结

合作为求解本文研究问题的算法依据，实验结果证明该算法在求解优度和求解效率上有一定的优势。本节主要介绍分支定界结合深度优先算法的详细过程和在求解二维矩形切割优化问题时的具体应用和改进。

3.1.1 基本思想

分支定界（BB）[9]是解决组合优化问题的常用算法。它由通过状态空间搜索到的候选解决方案枚举组成：候选解决方案的集合被认为是形成于根部具有完整集的有根树。由该算法探索并分支此树的一部分，表示解集的子集。在枚举分支候选解之前，针对最优解的上下估计边界检查分支，如果它不能产生比迄今为止算法发现的最佳解更好的解，则丢弃该分支。该算法依赖于搜索空间的区域或分支的下限和上限的有效估计。如果没有可用的边界，则算法退化为穷举搜索。

分支定界算法的目标是找到一个解 x，使得搜索空间或可行域 S 中的目标函数 $f(x)$ 的值最大或最小。本节中我们暂且假定需要最小化 $f(x)$；这个假设不失一般性，因为 $f(x)$ 的最大值可以通过最小值 $g(x) = -f(x)$得到，BB 算法主要更根据两个原则运行：

- 分支：BB 算法递归地将搜索空间分成更小的空间，然后最小化这些较小空间上的 $f(x)$；
- 剪枝：单独的分支就相当于对候选解决方案进行暴力枚举并对其进行全面测试。为了提高暴力搜索的性能，BB 算法跟踪它试图找到的最小值的边界，并使用这些边界对搜索空间进行“修剪”，消除不可能成为最佳解决方案的那些候选方案。

3.1.2 搜索流程

分支定界算法的主体框架如算法 3-1 所示。S_o 是通过启发式方法构造的初始解，B是当前初始解 S_o 对应的目标函数，表示到目前为止找到的最佳目标值，并将用作候选解的上界。如果没有找到合适的启发式方法来构造出初始解，建议使用深度优先搜索的变体，因为它可以快速生成完整解，作为上界。Q 是初始化队列用以保存部分解决方案，但不分配任何变量。通过遍历候选方案集合 Q，如果遍历到的当前节

点 n 已是一个完整的解决方案 S ，且它更优于之前的候选解的上界，则将 S 置为当前的最优解决方案，同时更新候选解的上界（第 6~8 行）。如果节点 n 不是完整的解决方案则对 n 进行分支，得到一系列分支集合

算法 3-1 **分支定界主体框架**

Input：S_o、B、Q

Output：S_{best}

```
1: S_o ← generate initial solution
2: B ← f(S_o)
3: Q ← initialize queue to hold a partial solution
4: while Q is not empty do
5:     take a node n of Q
6:     if n represent a single candidate solution S f(S) < B then
7:         S_best ← S
8:         B ← f(x)
9:     else
10:        branch on n to produce new nodes N
11:        for each n_i in N do
12:            if f(n_i) > B then do nothing
13:            else add n_i to Q
14:            end if
15:        end for
16:    end if
17: end while
18: return S_best
```

N，遍历 N，若 $f(n_i) > B$，则表明 n_i 的下界大于候选解的上界，因此再往下分支将永远不会得到最优解，没有意义，可以对该分支进行剪枝；否则将 n_i 加入候选解集合（第 9~16 行）。最后返回最优解S_{best}。

如图 3-1 给出了采用分支定界算法的初始情况和执行第一步操作的示例。这些访问节点序列 S 随着下一个待处理节点选择的策略变化而变化。如果下一个子问题的选择是基于子问题的边界值，那么在选择节点 S_1 和 S_4之后进行的第一个操作是分支，即将节点的解空间细分为两个或更多个子空间 S_{11}、S_{12}、S_{41}、S_{42} 在随后的

过程中继续进行访问。对于其中的每一个子节点检查子空间是否构成单个解决方案，在这种情况下，将其与当前最佳解决方案进行比较。否则，计算子空间的边界函数并与当前最佳解决方案进行比较。如果可以确定子空间 S_3、S_4 不可能包含最优解，则丢弃整个子空间（剪枝），否则将其与其他活的节点绑定在一起存储在活的节点池里。

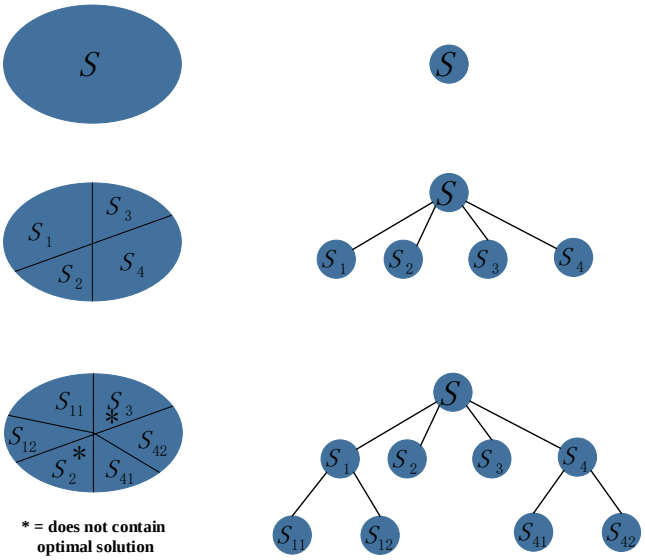


图 3-1 分支定界示例图

3.2 整体算法框架

本论文研究的二维矩形切割优化问题，采用迭代的带扰动机制和分支定界策略的启发式树搜索算法来求解。分支定界结合深度优先算法实现起来比较简单，是解决二维切割优化问题的一种快速的方法，为了提高解的优度和搜索效率，加入了迭代优化策略和禁忌策略来跳出无意义的搜索死循环，下面就详细描述求解二维矩形切割优化问题的启发式树搜索算法。

根据实际问题的规模，本文设计的迭代的带扰动机制和分支定界的启发式树搜索算法（Iterative Heuristic Tree Search Based On Perturbation，简称 IHTS-P），通过深度优先搜索算法指导原料的切割顺序和步骤，同时为了缩小搜索单个成品可能放置位置的范围使用分支和剪枝策略对求解过程进行加速，最后整个搜索流程完成后再通过迭代优化每块原料上的切割情况来求解二维矩形切割优化问题，求解算法的整体框架如算法 3-2 所示。具体地，算法首先会初始化，包括对输入数据的预处理和对

相关求解参数的配置（第 1~4 行）。接下来是一个主循环体，当总的运行时间没有超过给定上限时，进入循环体（第 5 行）。如果当前仍有物品的放置位置未分配，进入次循环体（第 6~11 行），首先会调用物品放置的序列函数，再调用启发式树搜索函数得到一个基于该物品的 $1-cut$ 最优解来指导该物品的放置，于此反复直到所有的物品的放置位置确定，接着更新剩余物品的数目（去掉已经得到最优放置的那部分物品，第 10 行）最后跳出循环。接下来若得到的当前解是一个完整的解（第 12 行），将它和历史最优解进行比较，若当前解优于历史最优解，则用该解更新当前最优解（第 9~11 行）。然后计算当前最优解的每块原料的利用率，找到当前原料利用率最低的那块原料，以此块原料起，重新构造从该原料到后续原料的解。为了避免多次重复启动都是从同一块原料重新开始计算，而使算法陷入单一选择，影响解决方案的质

算法 3-2 求解 TDRCO 问题迭代的带扰动机制和分支定界的启发式树搜索算法框架

Input： $Batch$、Bin、$Defect$

Output： S_{best}

1: initialization
2: $generation \leftarrow 0$
3: $S \leftarrow null$，$S_{best} \leftarrow null$ // initialize the solution S and optimal solution S_{best}
4: $items_{left} \leftarrow input.batch.size()$
5: **while** $!timer.isTimeOut()$ **do** // iterative heuristic tree search based on perturbation
6: **while** $items_{left} > 0$ **do** // if there are still items left unplaced
7: $PS_{best} \leftarrow null$ // initialize the partial best solution PS_{best}
8: $createItemBatchs(batch)$
9: $HeuristicTreeSearch(S, PS_{best})$ // get a 1-cut solution
10: $items_{left} \leftarrow items_{left} - PS_{best}.size()$
11: **end while**
12: **if** $S.size == items.size$ **then** // get a complete solution and use perturbation
13: **if** S is better than S_{best} **then**
14: $S_{best} \leftarrow S$
15: **end if**
16: $TabuBasedPerturbation(S, L, generation)$
17: **end if**
18: $generation \leftarrow generation + 1$
19: **end while**
20: return S_{best}

量，我们启动禁忌扰动机制。扰动机制中包括基于禁忌的扰动动作，执行禁忌扰动的时候，我们为当前利用率最低的原料加入禁忌步长来控制扰动的幅度，使得在经过多轮迭代搜索后，如果该块原料没有达到解禁要求，则不会重复作为新一轮优化的起始原料块（第 16 行）。扰动之后从利用率最低的那块原料开始，进行下一轮的启发式树搜索，更新迭代次数（第 18 行）。多次迭代后找到的最优解保存在 S_{best} ，当运行时间到达设置时间时整个迭代搜索停止，返回最优解（第 20 行），至此整个算法流程结束。

从以上的算法描述可以看出，IHTS-P 的主要算法框架包括：数据预处理、启发式树搜索：分支和剪枝策略、迭代优化和禁忌扰动机制。下文将详细介绍 IHTS-P 求解算法中核心搜索部分的内容，主要包含：解的表示、启发式树搜索算法：分支策略、剪枝策略、迭代优化解空间和禁忌扰动机制的详细内容。

3.3 解的表示

在 2.1.2 节中图 2-5 我们已经给出了成品完工顺序的深度优先切割树表示形式，而在本节算法中，如表 3-1 我们采用了一种 node 节点的结构体来存储成品的相关信息、在原料中的位置、切割顺序和保存了它在深度优先切割树表现形式中的位置信息来便于最后解的格式的转换。

表 3-1 node 结构体信息

成员	描述
$depth$	节点在树中的深度
$plate$	节点位于原料板的 id
$item$	放置的物品 id
$c1cpl$	当前左侧 $1-cut$ 位置（坐标）
$c1cpr$	当前右侧 $1-cut$ 位置（坐标）
$c2cpb$	当前下方 $2-cut$ 位置（坐标）
$c2cpu$	当前上方 $2-cut$ 位置（坐标）
$c3cp$	当前右侧 $3-cut$ 位置（坐标）
$c4cp$	当前上方 $4-cut$ 位置（坐标）
$cut1$	位于的 $1-cut$ id
$cut2$	位于的 $2-cut$ id
$flag$	高位在左，第一位代表物品是否旋转；第二位代表物品是否放置在瑕疵右方；第三位代表物品是否放置在瑕疵上方；第四位代表物品是否放置在虚拟桶 L4 中；第五位代表当前状态 $c2cpu$ 是否能够移动

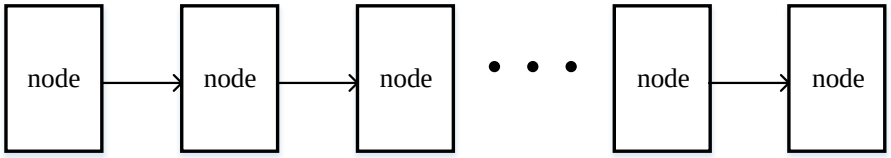


图 3-2 node 存储形式图

采用上表中的结构体形式存储解主要是便于解空间的索引和减少内存空间的占用。为了减少内存空间的占用，node 结构体只存储原料中切割出来的成品的相关信息，对于废料和余料在求解的过程中并没有存储下来，但是在最后解的格式转换阶段，我们可以通过已有成品的相关信息推出废料和余料的信息。为了便于解空间的索引，如图 3-2 我们可以维护一个 node 类型的列表，相对于树结构的存储形式它实现起来更加简单，并且能够进行快速地遍历、插入和删除操作。因此，若给定 n 个成品，在 IHTS-P 算法中只用申请 $O(n)$ 大小空间来存储解决方案，这对于提高求解效率，减少资源占用都有着显著的作用。

3.4 启发式树搜索

启发式树搜索模块的搜索范围是单件物品的放置（切割）位置，在下述算法 3-3 中我们已经给出了它的详细求解流程，该算法使用深度优先搜索算法并返回单个 $1-cut$ 的精确最优解。深度优先搜索模块首先对当前解、最优解、最优解的目标函数值、节点的深度和可分支节点集等变量进行初置，该问题的目标函数值 Obj_{best} 是每块原料的利用率，然后调用分支策略函数 $PartialBranch$，获得可分支的节点和当前部分可行解（第 6 行），接下来是一个主循环，如果当前的可分支节点数大于零，弹出可分支节点集$live_nodes$ 的尾部的节点 $node$。如果从当前节点继续分支得到的原料利用率的下界$usage_rate_lb$ 小于当前的目标函数值的最优解，则忽略掉当前分支节点 $node$，继续遍历后续节点，这就是我们在 3.4.2 将要介绍的剪枝策略（第 12~13 行）。如果当前节点是上一个节点的子节点，则继续向前遍历；如果当前节点所在深度和前一节点深度相等或更小，则需要回溯（第 14～16 行）消除额外的节点。将当前可行节点的数目赋值给$old_live_nodes_size$，继续调用基于该节点的分支策略函数并将返回值赋值给 $flag$（第 17~18 行）。如果完成了一次 $1-cut$ 分支，则更新当前的最优部分解和最优目标函数值（第 19~21 行）。将最终得到的最优部分解赋

值给 $solution$ 解集，并返回目标函数值的最优解供上层函数使用（第 23~24 行）。

算法 3-3 启发式树搜索算法框架

Input：$ResumePoint$、$batch$、$solution$

Output：Obj_{best}

1: $Obj_{best} \leftarrow -0.1$
2: $N_{live} \leftarrow null$ //the set of live nodes
3: $PS_{cur} \leftarrow null$ // current partial solution
4: $PS_{best} \leftarrow null$ // best partial solution
5: $pre_{depth} \leftarrow -1$ // the depth of previous node
6: $PartialBranch(ResumePoint, PS_{cur}, N_{live})$
7: while $N_{live}.size > 0$ **do**
8: $node \leftarrow live\_nodes.back()$
9: $usage\_rate\_lb = item\_area$ /
10: $((node.c1cpr - ResumePoint.c1cpl) * plateHeight)$
11: **if** $usage\_rate\_lb < best\_obj$ **then continue end if**
12: **if** $node.depth - pre\_depth == 1$ **then**
13: search forward
14: **else if** $node.depth - pre\_depth < 1$ **then**
15: search back
16: **end if**
17: $old\_live\_nodes\_size \leftarrow N_{live}.size$
18: $flag \leftarrow PartialBranch(node, PS_{cur}, N_{live})$
19: **if** $old\_live\_nodes\_size == N_{live}.size$ || $flag$ **then** // fill a complete 1-cut
20: **update** Obj_{best}、PS_{best}
21: **end if**
22: end while
23: $solution \leftarrow PS_{best}$
24: return Obj_{best}

我们在图 3-3 给出了该算法的示例化描述，其中物品 ID、放置位置和其他辅助数据结构存储在树节点中。其中蓝色的节点代表着当前物品可能放置的位置，红色弧线代表成品放置的先后顺序，它按照深度优先遍历。通过评估当前物品所有可能放置的位置，进而可以很方便地由一个节点确定下一个分支，而不是检查完整的解决方案，因此我们可以简单地将节点连接在一起来构建解决方案。

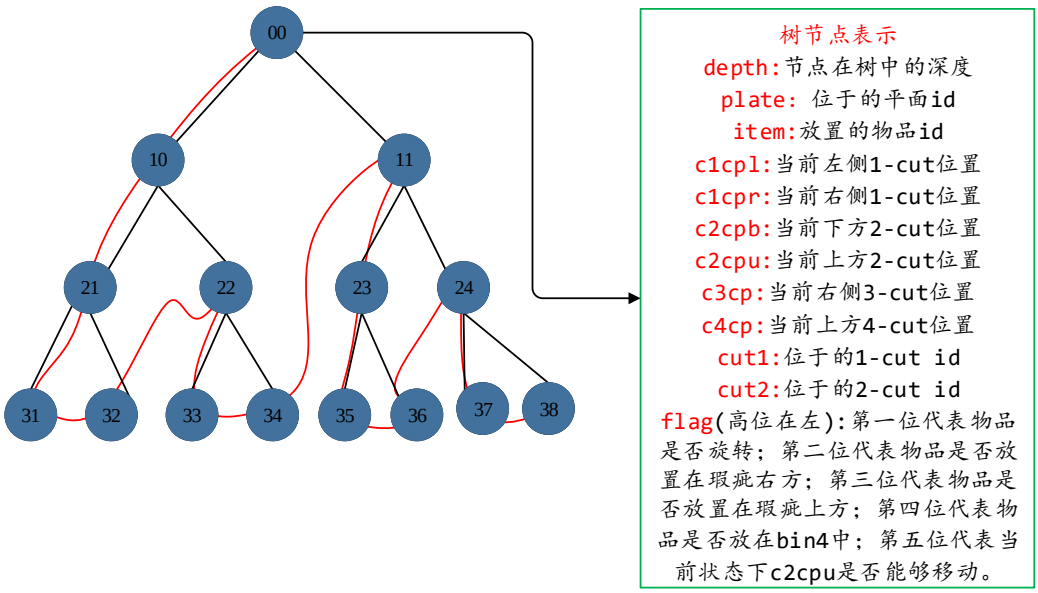


图 3-3 启发式树搜索示例图

3.4.1 分支策略

介绍详细的分支策略之前，基于本文研究的二维矩形切割优化问题需要建立一些基本原则。 首先，需要按照从左下到右上放置物品。其次，将物品放在切割位置或瑕疵旁侧。第三，一个物品可以水平或垂直放置，可以放置在缺陷的右侧或上侧。第四，只考虑分支过程中订单有序列表首部的成品。

在 3.1.1 小节我们已经对分支策略做了简单介绍，分支策略可以看作是通过添加约束或准则对搜索空间的一部分进行细分，通常以变量赋值的形式出现。如果所讨论的子空间被细分为两个，则成为二分分支，否则将称为多分分支。BB 算法是具有收敛性的，它确保每个生成的子问题的大小小于原始问题，并且确保原始问题的可行解的数量是有限的。另一点需要说明的是，通常认为生成的各子问题的是不相交的，以这种方式避免在搜索树的不同子空间中出现相同可行解的情况。

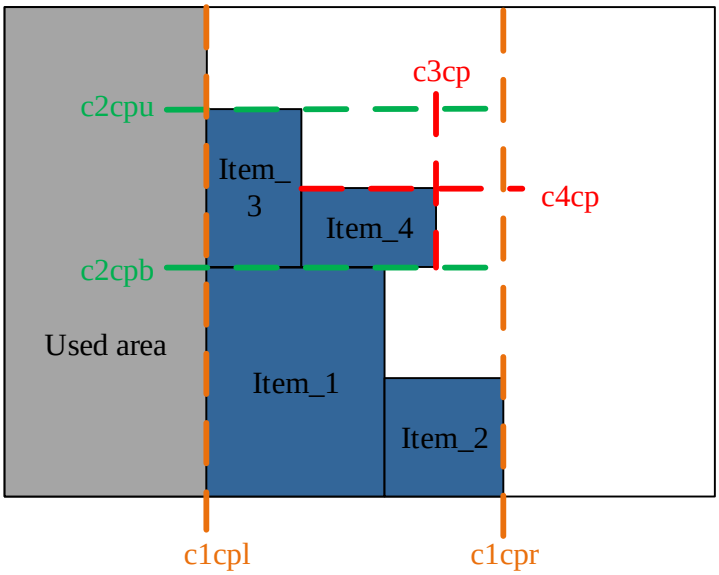


图 3-4 切割位置的直观含义图

为了表示分支的一个中间状态，我们定义整数变量 $c1cpl$（当前 $1-cut$ 位置左侧），$c1cpr$（当前 $1-cut$ 位置右侧），$c2cpb$（当前 $2-cut$ 位置底部），$c2cpu$（当前 $2-cut$ 位置顶部），$c3cp$（当前 $3-cut$ 位置）和 $c4cp$（当前 $4-cut$ 位置）。这些整数变量表示一块原料中的切割位置，如图 3-4 显示了它们的直观含义，其中蓝色矩形块为物品，灰色的矩形块是已经使用的原料，橙色的虚线表明的是 $c1cpl$ 和 $c1cpr$ 的实际含义，绿色虚线代表的是 $c2cpb$ 和 $c2cpu$ 的实际含义，红色的虚线表明的是 $c3cp$ 和 $c4cp$的实际含义。

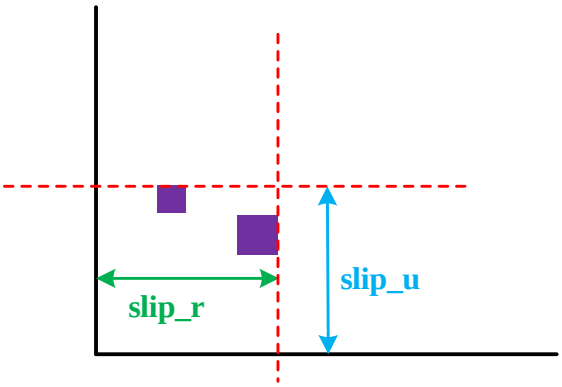


图 3-5 瑕疵位置描述图

如果放置物品时遇到瑕疵冲突，我们定义了 $slip\_r$ 代表最后一块与物品冲突的瑕疵右边界到左侧 cut 的距离，$slip\_u$ 代表最后一块与物品冲突的瑕疵上边界到下

侧 cut 的距离。若 $(slip\_r, slip\_u)$ 违反最小废料约束（见 2.2 节，切割模式相关的约束），则最小废料尺寸更新为 $(slip\_r, slip\_u)$ 的值，图 3-5 显示了瑕疵（紫色矩形块）位置的直观描述图。

算法 3-4 分支策略算法框架

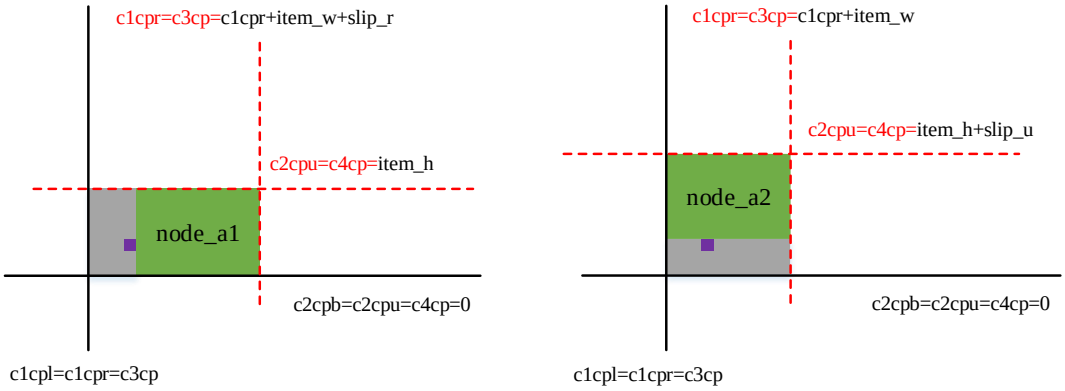
Input：$batch$、$solution$、$branch\_nodes$

Output：$branch\_nodes$

1: caseA: place item in a new L1
2: caseB: extend c1cpr or place item in a new L2
3: caseC: place item in the old L2 or a new L2 when item extend c1cpr too much

结合上诉我们定义的分支的中间状态，我们给出了分支策略算法的主要框架如算法 3-4 所示。在具体的分支过程中，我们考虑到当前的切割位置，将分支情况概括为三大类：

- Case A：$c2cpu$ 等于零，这是最简单的情况。物品放在 $c1cpl$ 和 $c2cpb$ 旁边或我们在 2.2.1 小节中提到的 L1 虚拟桶内。如果物品与瑕疵冲突，将其滑到瑕疵右侧或上侧。如图 3-6 Case A 的分支情况示例图，其中紫色方块代表瑕疵，绿色矩形代表物品的切割（放置）位置。

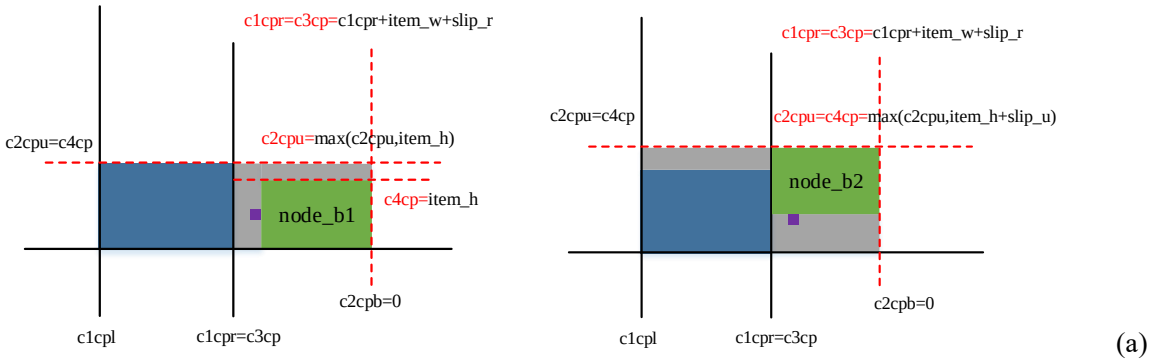


(a) 瑕疵右侧　　(b) 瑕疵上侧

图 3-6 Case A 分支情况示例图

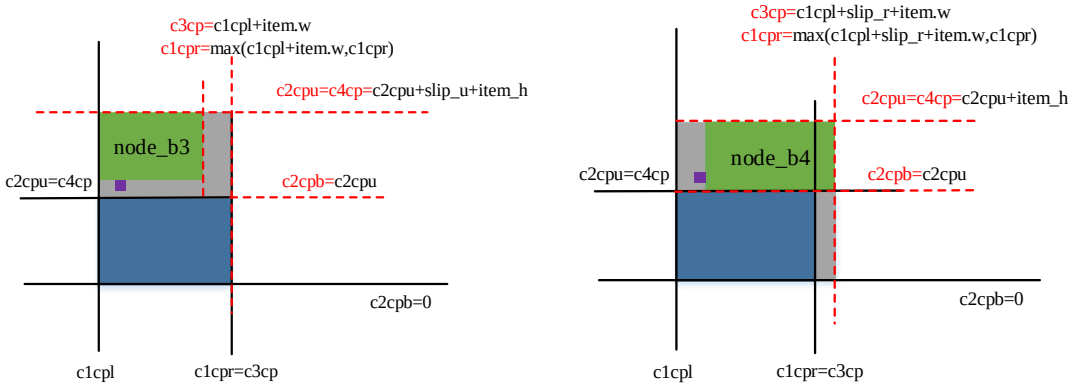
- Case B：$c2cpb$ 等于零且 $c3cp$ 等于 $c1cpr$。我们可以将一件物品放在 $c2cpb$ 和$c1cpr$ 旁，或者将它放在 $c2cpu$ 和 $c1cpr$ 旁。与 Case A 类似，如果物品与

瑕疵冲突，只需将其滑动到合适的位置即可。但如果放置的物品的上或右侧超过 $c2cpu/c1cpr$ 的值，我们应移动 $c2cpu/c1cpr$ 以确保在切割过程中不会切割到物品。如图 3-7 Case B 的分支情况示例图，其中图 (a) 、(b) 表示在有瑕疵冲突时，将物品分别放置在$c2cpb$ 和$c1cpr$ 旁侧的情况，



(a)

$c2cpb$ 和 $c1cpr$ 旁侧　　(b) $c2cpu$ 和 $c1cpr$ 旁侧，移动$c2cpu$



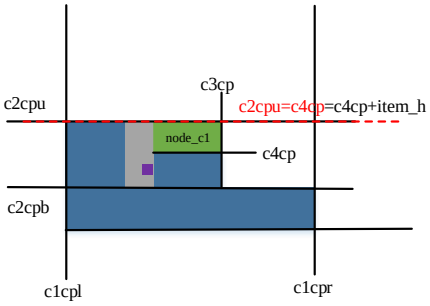
(c) $c2cpu$ 和 $c1cpr$ 旁侧　　(d) $c2cpu$ 和 $c1cpr$ 旁侧，移动 $c1cpr$

图 3-7 Case B 分支情况示例图

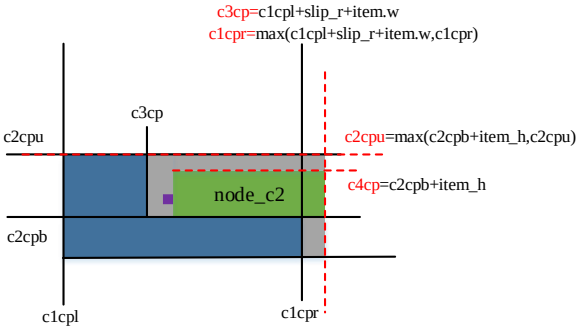
图 (b) 表示放置后上侧超过了 $c2cpu$，需要重新移动 $c2cpu$；图 (c) 、(d) 表示存在瑕疵冲突时，将物品分别放置在 $c2cpu$ 和 $c1cpr$ 旁侧，其中图 (d) 表示放置后右侧超过了 $c1cpr$，应当重新调整 $c1cpr$。

- Case C：$c2cpb$不等于零。物品可以放在 $c2cpb$ 和 $c3cp$ 旁侧或 $c2cpu$ 和 $c1cpl$ 旁侧或 $c3cp$ 和 $c4cp$ 旁侧。如果物品位于 $c3cp$ 和 $c4cp$ 旁边，则其上限应等于 $c2cpu$，其右边界应等于 $c3cp$。如图 3-8 Case C 的分支情况示例图，其中图 (a)、(b) 和图(c) 表示存在瑕疵冲突时，物品分别放置在$c2cpb$ 和$c1cpr$ 旁侧的情况，图 (b) 表示放置物品后右侧超过$c1cpr$ ，应当重新调整 $c1cpr$，而图 (c)

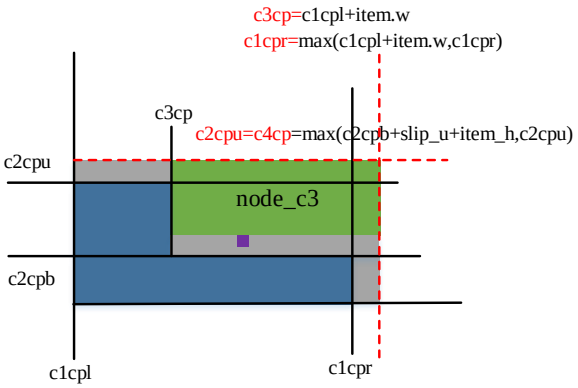
表示放置物品后上侧和右侧超过了 $c2cpu$ 和 $c1cpr$，应当移动 $c2cpu$ 和 $c1cpr$；图 (d) 和图 (e) 表示存在瑕疵冲突时，将物品放置在 $c2cpu$ 和 $c1cpl$ 旁侧，根据瑕疵位置将其滑到瑕疵的右侧或上侧；图 (f) 和图 (g) 表示在处理瑕疵冲突时，将物品放置在 $c3cp$ 和 $c4cp$ 旁侧，按照约定准则位于瑕疵的上侧或右侧。



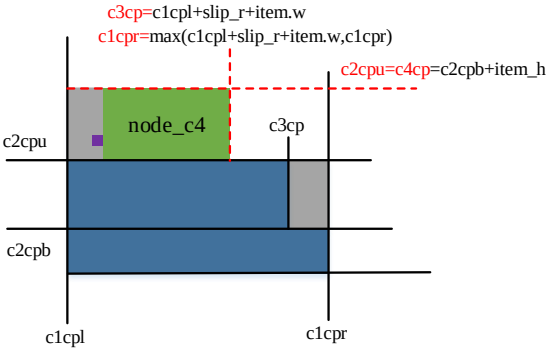
(a) $c2cpb$ 和 $c3cp$ 旁侧



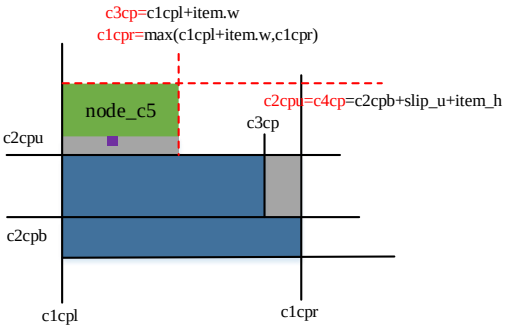
(b) $c2cpb$ 和 $c3cp$ 旁侧，移动 $c1cpr$



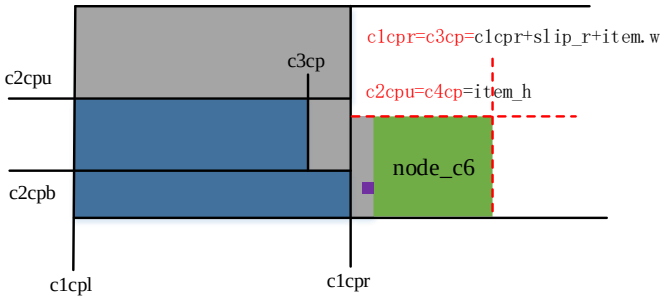
(c) $c2cpb$ 和 $c3cp$ 旁侧，移动 $c1cpr$ 和 $c2cpu$



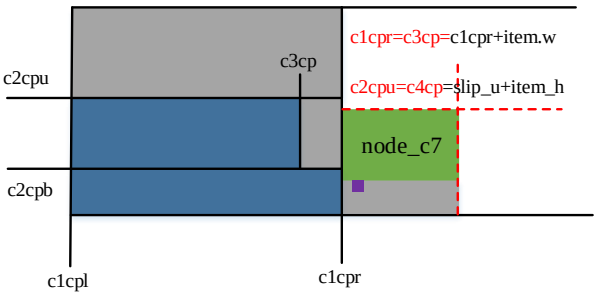
(d) $c2cpu$ 和 $c1cpl$ 旁侧，瑕疵右侧



(e) $c2cpu$ 和 $c1cpl$ 旁侧，瑕疵上侧



(f) $c3cp$ 和 $c4cp$ 旁侧，瑕疵右侧



(g) $c3cp$ 和 $c4cp$ 旁侧，瑕疵上侧

图 3-8 Case C 分支情况示例图

简单地将一个成品放在当前切割位置或瑕疵旁侧可能会违反某些约束。因此，我们需要检查废料区域大小，$1-cut$ 宽度和 $2-cut$ 高度是否在最小废料范围内或某切割位置是否经过每个分支节点中的某些瑕疵。因此，如果发现一个分支节点违反了某些约束，我们将尝试通过移动切割位置来修复它或直接放弃修复该分支节点。

3.4.2 剪枝策略

在 3.4 小节中，我们已经简单地介绍了剪枝策略，如算法 3-3 第 10~11 行所示。为了加快搜索过程，我们对搜索树采用了一些剪枝策略。每个节点的下限将在分支之前进行评估，如果节点的下限比当前已有的最优目标值更差，我们将去掉此分支。下限由公式(3-1)和(3-2)计算，其中 lb 表示下限，ua 表示已使用的原料面积，la 表示左侧物品面积，PH 表示原料块高度。

$$lb = ua/(c1cpr + lw) * PH \tag{3-1}$$

$$lw = (la - (c2cpu - c2cpb) * (c1cpr - c3cp) - (ph - c2cpu) * (c1cpr - c1cpl)) / PH \tag{3-2}$$

3.5 迭代优化

通常仅运行一次启发式树搜索算法来获得最优的解决方案是相当困难的，尤其是当测试样例的规模非常具有挑战性时。在这种情况下，一个容易实现但不够有效的策略是简单地从随机初始配置中重新启动新一轮启发式树搜索。然而尽管我们运行一次启发式树搜索算法得到的解不是最优解，但它通常具有相当一部分的价值，如果我们通过扰动在下轮计算时跳过那些比较差的解而不是每一轮都完全抛弃上一轮计算的结果重新计算，则不仅可以提高求解效率，还可以引导启发式树搜索向更有潜力的解空间搜索。

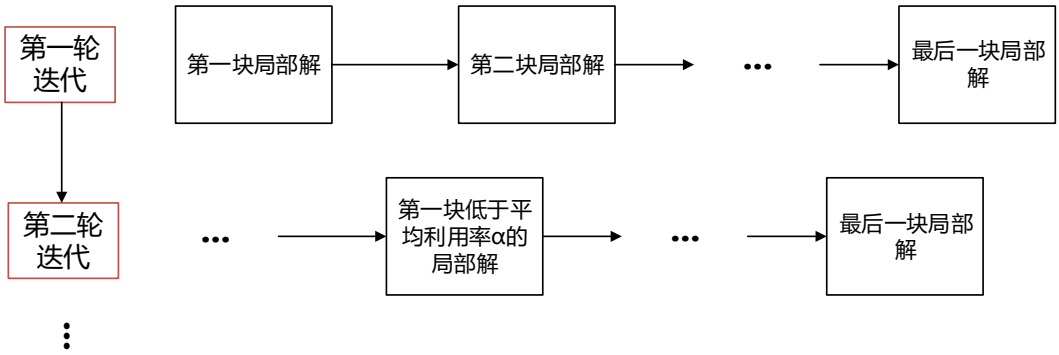


图 3-9 迭代优化示意图

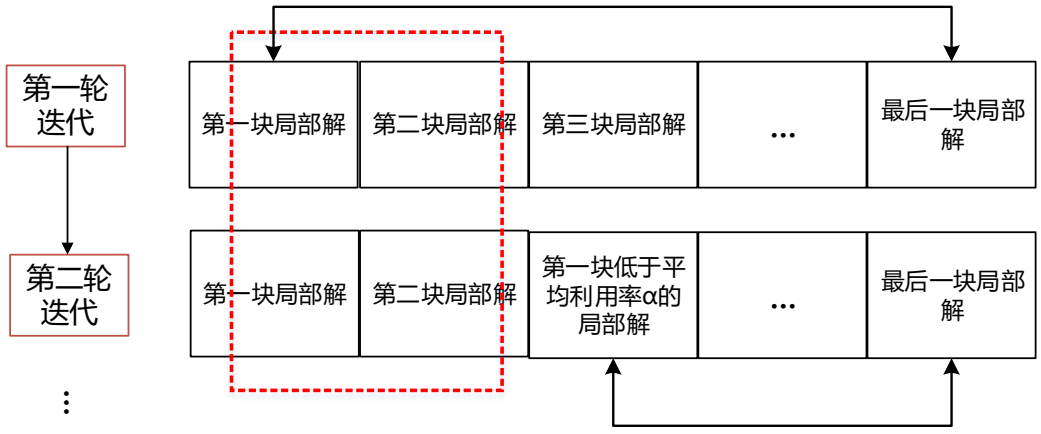


图 3-10 迭代优化解的示意图

基于上述想法我们提出了一种基于最差解迭代优化的方案，如图 3-9 所示，在每轮迭代过程中我们仅挑选当前迭代过程中利用率最低的那块原料，我们认为该原

料块的解是最差的。而由于原料的切割问题是相互牵连的，后一块原料的切割方案会依赖前一块原料的切割模式。因为若前一块原料切割完成了物品 A，那么后一块原料就不需要考虑切割同样规模的物品。如果我们仅仅重复计算利用率最低的那块原料，可能会导致解的结构被破坏而使原有的解变成不可行解。一种简单且有效的做法是以该块原料为起点（启动原料块）将包括其自身和后续的其他所有原料块的切割方案均重置，再以利用率最低的那块原料为起始原料块调用下一轮的启发式树搜索算法求解原问题中原料块集合的子集的切割方案即所谓的原问题的部分解。如图 3-10 每轮迭代中解的构成情况，第一轮迭代，调用启发式树搜索算法，求出完整的解决方案；第二轮迭代，保留启动原料块前的每块原料的上一轮切割方案，从启动原料块开始重新计算后续原料块的切割方案。后续迭代中解的构成类似，直到达到运行时间上限迭代优化终止，输出当前最优解。

3.6 扰动机制

可以看出本问题中设计的迭代优化策略对于提升解的质量和保证求解效率都有一定的效果，但它也存在一定的设计弊端，就是启动原料块的选择我们只依赖于当前解的评估结果而没有考虑到历史因素，因此我们可能会出现多次从同一原料块启动，这对于解的疏散性是不利的。因此基于已有的迭代优化策略再结合本文研究的具体问题我们提出了一种基于禁忌思想的扰动机制。禁忌的思想是源自于启发式方式中的禁忌搜索，它和局部搜索类似都是在搜索过程中迭代地从一个潜在的解决方案通过邻域搜索来找到局部最优解替换当前解，但区别在于禁忌搜索还会维护一张禁忌表用来记录已经执行过的动作，在进行下一个动作前，会查询该动作是否已存在于禁忌表中，若在且当前迭代次数小于禁忌步长，则该动作被禁止执行直到达到解禁要求后方可再执行。由于禁忌表的存在使得算法不会在短时间内重复执行相同的动作，增强了解的疏散性。

在前文 3.2 小节中算法 3-2 中我们已经简要介绍了本文中的扰动机制。为了提高解的优度，我们通过迭代调用启发式树搜索算法来优化解，而为避免出现多次从同一块原料启动优化，我们引入了禁忌的思想。将已经选过作为启动原料块的原料加入

算法 3-5 禁忌扰动算法框架

Input：$solution$，$generation$

Output：$wrost\_plate\_id$

```
1: wrost_usage_rate ← 1.0, wrost_plate_id ← 0
2: getPlatesUsageRate(plate_usage_rate, solution)
3: if tabu_restart_plate.size() < plate_usage_rate.size() do
4:     tabu_restart_plate.resize(plate_usage_rate.size(), generation)
5: end if
6: for each plate index i in plateIndex // find the wrost no tabu plate.
7:     if generation ≥ tabu_restart_plate[i] & plate_usage_rate[i] <
         wrost_usage_rate
8:         wrost_plate_id ← i
9:         wrost_usage_rate ← plate_usage_rate[i]
10:    end if
11: end for
12: tabu_restart_plate[wrost_plate_id] ← α1 · generation + α2 · rand(L)
```

禁忌表，设置禁忌步长，在禁忌步长内，该原料不会被重复选作启动块原料。具体如算法 3-5 所示，算法经过一轮的启发式树搜索得到一个完整解，然后启动扰动机制。在该部分算法流程中，首先调用函数$getPlatesUsageRate$ 计算每块原料的利用率（第 2 行），然后初置禁忌表 $tabu\_restart\_plate$ 将每块原料的初始禁忌长度设置为当前的迭代次数（第 3~5 行）。接下来就是算法的核心部分，遍历整个原料块集合查找没有被禁忌的利用率最低的原料块，然后将它的保存，作为迭代优化的起始原料块（第 6~11 行），这样我们不仅考虑了当前的最差解，同时又避免了只考虑本轮搜索而忽略历史情况造成的“短视”结果。最后就是按照禁忌步长的设定依据见公式(3-3)，为利用率最低的原料块重置禁忌步长（第 12 行）。

$$L = \alpha_1 \cdot threshold + \alpha_2 \cdot random(threshold) \tag{3-3}$$

一般来说禁忌步长是和邻域空间大小成强相关的，在本文中我们可以象征性地认为领域空间是以原料块为单位的。禁忌步长一般是由两个部分组成的，一部分是搜索过程中的各个动作的距离限制，对于某一轮搜索来说它是一个固定值；另一部分是一个随机值，大小不能超过限制距离的大小，这两部分呈线性相关，通过参数 α_1 和

α_2 来调整这两部分的比重最终组成总的禁忌步长。对应到本文研究的问题中，为了简化操作，我们将固定值设置为当前的迭代次数，而限制距离则是依赖于当前解中已使用的原料块的总数来制定。

3.7 本章小结

本章首先介绍了启发式树搜索算法，描述分支定界算法的基本思想和搜索流程。IHTS-P 采用了启发式树搜索算法为主体搜索框架，以深度优先搜索算法来获得单个物品的切割位置，同时引入分支策略来添加问题中的约束和对搜索空间进行细分，对应提出的剪枝策略是为了在分支策略进行之前对分支进行评估，对于潜力不大的分支进行剪切来提高求解效率。另外本文还提出了迭代优化策略和和基于禁忌的扰动机制，旨在保证解空间疏散性的前提下对解进行优化。

4 实验结果分析

在前面的章节中，针对二维矩形切割优化问题，已经详细描述了求解该问题的迭代的带扰动机制和分支定界的启发式树搜索算法。在本章节中，通过对算例进行分析，将 IHTS-P 算法通过算例测试得到的切割方案，和模型的计算结果作对比，从最优性和时间效率两个方面比较本文提出的两种模型算法和 IHTS-P 算法的计算性能。

4.1 测试方案设计

为了更好地衡量算法的求解性能，需要在算例集上对算法进行验证，本节将主要描述测试算例的规模和类型，以及实验平台的各类配置参数。

4.1.1 测试算例

本文研究中涉及到的算例集来源于浮法玻璃厂商 Saint-Gobain Glass France（SGGF）提供的真实算例的简化[1]。算例集 A 包含小、中规模的算例共 20 例，而算例集 B 包含常规大小（大规模）的算例共 20 例。

算例包含所有必需数据的输入，文件以 CSV 格式给出。具体在文件中以“;”作为分隔符，第一行数据始终包含标题（列名）。例如，在算例集 A 或 B 中，一个问题实例由两个文件组成：成品文件和瑕疵文件。第一个文件存储物品相关的数据，第二个文件存储玻璃板上瑕疵相关的数据。下面给出每个文件及其结构的详细描述。在本小节的剩余部分中，使用了一个样例 A0 作为示例。

表 4-1 算例 A0 成品文件

成品编号	长度	宽度	订单列表集	次序
0	600	600	0	1
1	600	600	0	2
2	1242	247	0	3

如表 4-1 所示，成品文件包含一系列需要被切割的成品。每一行（除了标题）代表一件成品的相关数据：成品的编号、成品的尺寸：高度和宽度、成品所在的订单列

[1] http://www.roadef.org/challenge/2018/en/instances.php

表集编号、成品在订单列表集中的访问先后次序（从 1~|订单列表集|）。值得注意的是：由于成品在放置（切割）的过程中允许旋转，因此如何考虑成品的尺寸并不重要。

表 4-2 算例 A0 瑕疵文件

瑕疵编号	原料编号	X 值	Y 值	宽度	高度
0	0	50	50	3	3
1	0	3	3207	3	3
2	3	1090	1685	2	4

如表 4-2 所示，瑕疵文件包含所有原料上的瑕疵信息。每一行（除了标题）代表了一件瑕疵的相关数据：瑕疵编号、瑕疵对应的原料的编号、瑕疵的方位信息：位于原料块左下角的 x 和 y 坐标、瑕疵的尺寸：宽度和高度。

4.1.2 实验环境

本文算法的实现均是在个人 PC 电脑上完成，具体的硬件配置和软件环境如表 4-3 所示。其中模型和启发式算法都是在同一 IDE 下编程实现，数学规划求解器用于辅助模型算法的实现。

表 4-3 测试环境配置表

类型	名称	配置或参数
硬件环境	处理器	Intel(R) Core (TM) i3-3240 CPU@ 3.40 GHz
	内存	8GB
	硬盘	500GB
软件环境	操作系统	Microsoft Windows 10 专业版 (64 位)
	IDE	Visual Studio 2017
	求解器	Gurobi 8.0

4.1.3 测试结果说明和可视化展示

算例集 A 或 B 中的某一个算例 x 对应的解决方案的名称必须命名为 $Ax\_solution.csv$。在上文 3.3 节中我们提到解决方案是按照深度优先切割树表示，它是由森林组成，其中每种切割模式代表着森林中的树，如图 4-1 所示。对应到表 4-4

的算例 A0 的解文件中，其中每一行必须对应于树中的一个分支节点。节点始终是一个矩形，它可以是原料，分支（带子节点的节点），成品，废料或余料。在表中我们为每个节点提供以下信息：原料编号、节点编号、节点左下角的 x 坐标、节点左下角的 y 坐标、沿节点 x 坐标轴的宽度、沿节点 y 坐标轴的高度、节点种类（值大于等于 0 表示成品编号，值等于-1 表示废料，值等于-2 表示分支节点，值等于-3 表示余料）、切割顺序（值等于 0 表示原料、值等于 $\alpha \in \{1,2,3,4\}$，表示 $\alpha - cut$）、父节点（如果当前节点为根节点，该值为空）。

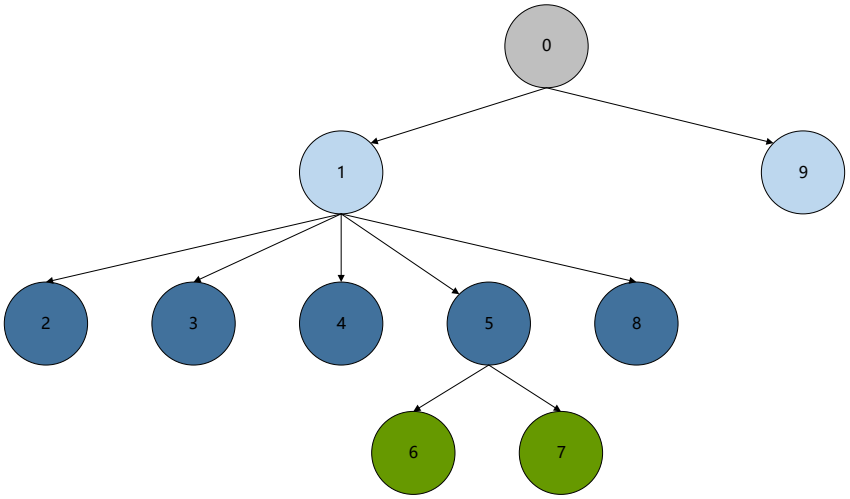


图 4-1 切割模式示意图

表 4-4 算例 A0 解文件

原料编号	节点编号	X 值	Y 值	宽度	高度	种类	切割顺序	父节点
0	0	0	0	6000	3210	-2	0	
0	1	0	0	600	3210	-2	1	0
0	2	0	0	600	53	-1	2	1
0	3	0	53	600	600	1	2	1
0	4	0	653	600	600	1	2	1
0	5	0	1253	600	1242	-2	2	1
0	6	0	1253	247	1242	2	3	5
0	7	247	1253	353	1242	-1	3	5
0	8	0	2495	600	715	-1	2	1
0	9	600	0	5400	3210	-3	1	0

得到解决方案文件后，使用 checker 程序来对解决方案的可行性进行测试，包括：解析算例文件的可行性、算例解决方案的可行性、计算目标函数的值和一些其他数据

（使用的原料块数量、余料的大小等）。当解决方案可行性满足后，生成切割方案的可视化示意图，如图 4-2 所示。

图 4-2 A0 算例可视化解决方案

4.2 测试算例分析

算例分析是指使用适当的统计分析方法对算例数据进行分析，提取有用信息和形成结论而对算例加以详细研究和概括总结的过程[45]。它类似于我们实现算法中的第一步：数据预处理，对原始算例进行标准化处理包括数据的审核、数据的筛选和数据的排序等[46]。往往一份有效的算例分析结果会为我们后续设计算法和算法的实现提供很大的帮助，特别是我们所研究的二维矩形切割优化问题，它源自于真实的工业界的问题，往往没有普适性算法可以直接用来求解该问题，所以我们可以通过分析算例，来找到算法实现的突破口。

在本文研究中我们分析了算例集 A 和 B 中的每一个算例基本信息的分布情况以表格的形式展现。如表 4-5 所示，包括：订单（成品）总数、有序订单列表数、瑕疵总数和每块原料上的瑕疵数、订单（成品）总面积、瑕疵总面积、原料块数下界、原料块数上界、订单长宽分布、订单列表大小分布、瑕疵长宽分布、尺寸相同的物品数（尤其是在同一个订单列表中连续出现的）等。

对于尺寸相同的物品数目，除了给出它的分布情况表，我们还给出了表格对应的气泡分布图（图 4-3）来便于直观的分析。图中横纵坐标分别代表物品的宽度和高度。

表 4-5 算例分析表

描述	分布	有序列表编号	成品总数	成品宽度	成品高度	有序列表编号
成品总数	5	0	5	581	276	0
瑕疵总数	29			1396	781	0
成品总面积	4514704			1426	648	0
瑕疵总面积	2142			1550	738	0
原料块数上界	1			1578	758	0
原料块数下界	2					

成品宽度	成品高度	成品数目	有序订单列表编号 0			原料块编号	瑕疵数目
			成品宽度	成品高度	成品数目		
581	276	1	1578	758	1	0	3
1396	781	1	1550	738	1	1	1
1426	648	1	581	276	1	2	2
1550	738	1	1396	781	1	3	0
1578	758	1	1426	648	1	4	0

瑕疵宽度	瑕疵高度	原料块编号
1	1	0
1	1	0
1	1	1
1	1	2
1	2	2
2	1	0

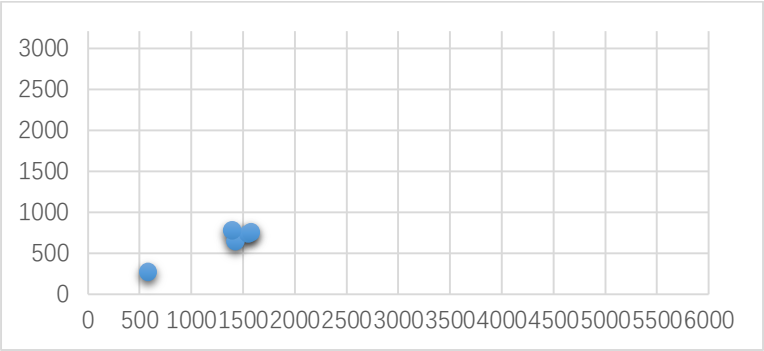


图 4-3 物品尺寸气泡分布图

原料块数下界我们采用了两种方式来估算：一种是用成品总面积比单块原料面积估算如公式(4-1)所示，这种处理方式实现起来非常简单，但是结果相对来说比较粗糙；一种是用一维装箱或背包问题估算，这种方式实现起来复杂一些，但是结果更加接近最终的解，便于解空间的缩小。

$$lowerBoundOfPlateNum = TotalAreaOfItems \,/\, AreaOfPlate \tag{4-1}$$

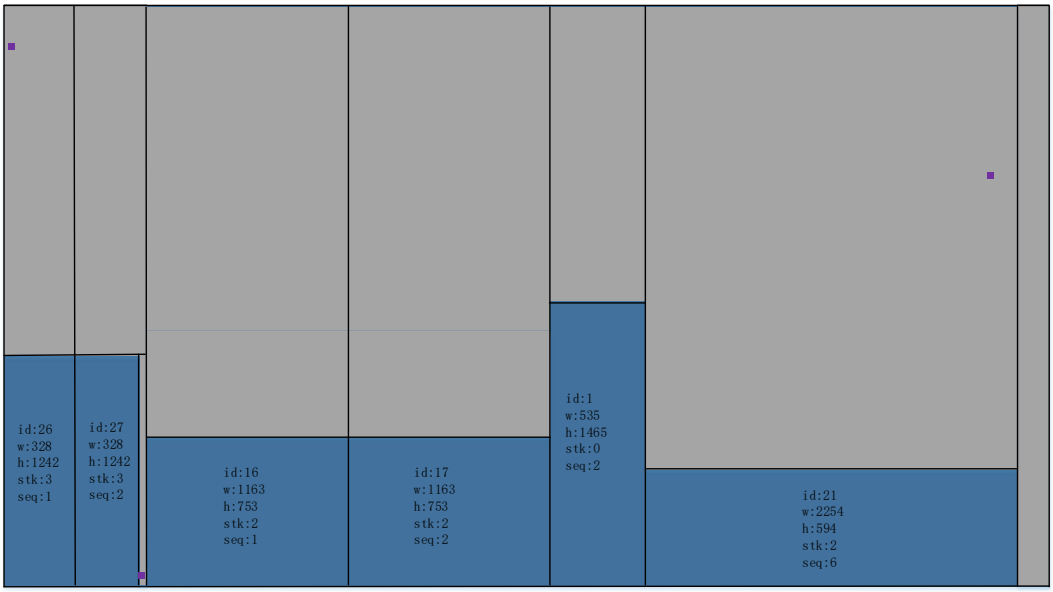


图 4-4 原料块数上界评估方式示意图

对于上界的估计我们采用的是一种比较简单的做法，将成品平铺于原料块的底层，我们只考虑原料块的使用宽度。当成品放置在原料块上，剩余的可用原料块宽度等于去掉当前已放置（切割）成品的宽度，如果剩余的原料块宽度无法满足将要切割的成品尺寸，则重新选择下一块原料进行成品的放置（切割），如上图 4-4 给出了原料块上界评估中其中一块原料块的放置（切割）方式。

4.3 参数设定

参数文件对所有的算例集是通用的，包含一些固定的优化参数或约束值：原料块的使用上限、原料块的尺寸标准化设置、切割步骤中的相应参数约定、瑕疵的尺寸约束、随机种子生成约定、运行时间阈值、模型中的迭代次数阈值和禁忌扰动机制中相关参数。该文件内容的具体描述和总结如下表 4-6 所示。由于原料块都是标准尺寸，

因此在实现过程中不需要再额外添加对原料块的相关说明；原料块的编号固定为 $0,1,2,3,\ldots,nPlates-1$，必须按照顺序使用；随机种子的生成对于每种算法每个算例都必须按照同一方式生成；由于扰动机制中涉及到的距离阈值与原料块数成正相关，而原料块数的使用上限是 100，因此距离阈值的大小对于算法的求解效率影响不大，我们直接将其设置为邻域空间的大小 n，对于运行时间的设置我们设置了两组数值：3 min 和 60 min，对于 0 小节我们将具体分析对于同一算例在不同运行时间下的解的质量。

表 4-6 参数文件

名称	数值	说明
$nPlates$	100	可用原料块的上限值
$widthPlates$	6000	原料块的固定宽度
$heightPlates$	3210	原料块的固定高度
$min1Cut$	100	两次连续 $1-cut$ 之间的最小距离（瑕疵除外）
$max1Cut$	3500	两次连续 $1-cut$ 之间的最大距离（余料除外）
$min2Cut$	100	两次连续 $2-cut$ 之间的最小距离（瑕疵除外）
$minWaste$	20	瑕疵的最小宽度和高度
$randSeed$	统一	随机种子
$timeout$	180、3600	运行时间阈值
$maxIter$	1000000000	最大迭代次数
$threshold$	n	距离阈值
α_1	1	禁忌扰动中禁忌长度固定值部分系数
α_2	1	禁忌扰动中禁忌长度随机值部分系数

4.4 实验结果对比

二维矩形切割优化问题在工业界有着广泛的应用，吸引着国内外众多科研工作者投身其中寻求不同的算法来求解它。本节主要对比模型算法中的两种不同模型的求解结果，接着将模型算法与我们所设计的 IHTS-P 算法进行比较，分析数学规划和启发式方法在求解该问题上的优劣性，最后我们就求解质量和求解效率来分别分析模型和 IHTS-P 算法，包括和当前的最优解进行对比。

4.4.1 CM 算法与 IM 算法对比

本文中提出的数学规划算法：完整模型（CM）和迭代模型（IM）算法（2.2 小节）都可以单独求解二维矩形切割优化问题，其中完整模型是最先提出来的，求解效率和求解结果稍逊色；迭代模型是基于完整模型的基础上提出来的，求解结果和求解效率相对于完整模型算法较好。本节主要比较本文提出的迭代模型算法和完整模型算法的求解效果以及迭代模型算法相较于完整模型算法的求解效果。

表 4-7 中，给出了完整模型和迭代模型结果对比。附录I表格 2 中统计了 CM 和 IM 模型算法在算例集 A、B 中每种算例运行时间为 1 小时的 5 次计算结果的平均值。由于完整模型求解时间较长，在较短的时间（3min）内无法求解出合法解（Time

表 4-7 完整模型与迭代模型对比——改进幅度

A Set			B Set		
ID	IN	Avg.(%imp)	ID	IN	Avg.(%imp)
A1	5	0.00	B1	68	**+19.55**
A20	17	**+13.04**	B20	124	**+100.00**
A17	21	-12.96	B6	204	**+11.94**
A6	37	**+39.80**	B5	207	**+54.02**
A16	38	-4.70	B10	214	-11.59
A19	47	-28.46	B7	241	**+20.24**
A12	50	**+37.27**	B9	247	-44.71
A7	57	**+38.38**	B17	247	**+41.58**
A9	63	**+11.05**	B18	258	**+100.00**
A3	68	**+27.90**	B4	261	**+24.15**
A4	68	**+7.71**	B14	267	**+5.29**
A2	72	**+25.88**	B11	274	-31.61
A18	73	**+36.85**	B16	300	**+3.42**
A10	86	**+21.75**	B3	332	**+15.46**
A11	86	-0.83	B8	334	**+12.60**
A5	97	**+13.56**	B19	371	**+100.00**
A8	129	-7.05	B2	383	**+10.26**
A13	272	**+14.91**	B15	431	**+100.00**
A14	361	**+19.26**	B12	439	**+21.84**
A15	392	**+3.90**	B13	656	**+100.00**

$$\%imp = (Obj_{CM} - Obj_{IM})/Obj_{CM} * 100 \quad (4\text{-}2)$$

Out），因此为了比较 CM 和 IM 算法的性能，减少测试环境和算法的偶然性因素，对于算例集 A、B 中不同类型、不同规模的每种算例均计算 5 次，每次的运行时间设定为 60min，并比较平均（Avg.）的计算结果改进幅度，算例按照物品的规模（Item Number, IN）升序排列如表 4-7 所示。迭代模型相对于完整模型的改进幅度 $\%imp$ 计算方式如公式(4-2) 所示，其中 Obj_{CM} 代表完整模型的目标函数值， Obj_{IM}代表迭代模型的目标函数值。从表中可见，在绝大多说情况下，IM 算法相对于 CM 算法的改进幅度值为正（表 4-7 中加粗部分），说明本文提出的 IM 算法相对 CM 算法有改进，但针对较小规模和中等规模的算例（A19，B9，B11 等），CM 算法的计算结果比 IM 算法有优势，针对较大规模算例（IN＞300），IM 算法相对于 CM 都有改进，甚至有些改进是 100%，说明 IM 算法此时具有绝对的优势。这是因为 CM 算法采用的是完整模型，它将所有的约束、目标和决策变量一次性加载到模型中，所以预处理就需要消耗大量的时间，所以该算法对于较短时间内求解大规模算例没有优势，甚至出现超时无解的情况，但是在求解小规模算例在求解效果上有一定优势。而 IM 模型采用迭代的方式每次加入部分约束和决策变量，使得模型的规模急剧缩小，所以它能在较短时间内求解出大规模算例，且在相对充足的时间内，它的求解性能依然相对于 CM 有较大优势。

表 4-8 同样给出了 CM 和 IM 算法的计算结果对比，比较的是 IM 算法相较于 CM 算法改进算例的数量相对于测试算例总量的占比。我们分别统计了算例集 A、B 中的情况如表所示。列（＜）表示算法 IM 的求解结果小于算法 CM，即 IM 算法相对于 CM 算法有改进，列（＝）表示算法 IM 和 CM 的的求解结果，即 IM 算法和 CM 算法的改进效果相当，列（＞）表示算法 IM 的求解结果大于算法 CM，即 IM 算法相对于 CM 算法的改进为负。在表 4-8 我们可以看出，对于算例集 A、B 中，IM 算法 在绝大多数情况下相对于 CM 算法是有改进的符合我们上表的分析。具体地，对于算例集 A，IM 算法的改进算例占比为 70%，对于算例集 B 的改进占比为 85%，改进幅度超过 2/3，这得益于我们对完整模型的改进，将问题进行细分，将约束和决策变量分批加入模型中，多次迭代求解，提高求解效率同时对于小规模问题充分发挥求

解器的求解性能优势；在极少数情况下（5.00%和 0.00%），IM 算法和 CM 算法的改进幅度相当；在一定比例下（算例集 A 中 25.00%，算例集 B 中 15%），IM 算法相对于 CM 算法有些许弱势，这主要归因于 CM 算法在求解小规模算例上有一定优势。

表 4-8 完整模型与迭代模型对比——改进算例占比

Criteria	Set	%		
		<	=	>
Avg.	A	70.00	5.00	25.00
	B	85.00	0.00	15.00

4.4.2 IHTS-P 算法与 CM 算法对比

表 4-9 IHTS-P 算法与 CM 算法结果对比——改进幅度

A Set			B Set		
ID	IN	Avg.(%imp)	ID	IN	Avg.(%imp)
A1	5	0.00	B1	68	**+48.56**
A20	17	**+20.01**	B20	124	**+100.00**
A17	21	0.00	B6	204	**+40.77**
A6	37	**+50.20**	B5	207	**+54.02**
A16	38	**+31.96**	B10	214	**+16.42**
A19	47	**+15.95**	B7	241	**+39.60**
A12	50	**+56.84**	B9	247	**+13.47**
A7	57	**+65.02**	B17	247	**+52.54**
A9	63	**+45.91**	B18	258	**+100.00**
A3	68	**+51.63**	B4	261	**+53.62**
A4	68	**+27.17**	B14	267	**+43.10**
A2	72	**+25.88**	B11	274	**+24.32**
A18	73	**+51.02**	B16	300	**+42.00**
A10	86	**+49.73**	B3	332	**+37.66**
A11	86	**+42.55**	B8	334	**+45.33**
A5	97	**+51.73**	B19	371	**+100.00**
A8	129	**+44.56**	B2	383	**+37.76**
A13	272	**+48.77**	B15	431	**+100.00**
A14	361	**+51.41**	B12	439	**+43.87**
A15	392	**+38.23**	B13	656	**+100.00**

在上一小节我们对比了两种模型算法：CM 和 IM 算法的求解效果，其中 CM 算法的求解效果相对于 IM 算法逊色。之后我们又提出了 IHTS-P 算法， IHTS-P 算法是基于启发式树搜索的算法。本节主要比较 IHTS-P 算法相对于 CM 算法在求解质量上的改进幅度。

表 4-9 给出了本文中提出的 IHTS-P 算法相对于 CM 算法的改进幅度。由于 CM 算法的求解性能限制，我们依然选择了 A、B 算例集中的共 40 组算例，单次运行时间为 1 小时共计算 5 次，统计 5 次中平均（Avg.）改进幅度 $\%imp$，其中 IHTS-P 算法相对于 CM 算法的改进幅度的求解方式类似于公式(4-2)。具体的平均求解结果见附录I中的表格 2。由表 4-9 中可以看出，在几乎所有算例测试结果中，IHTS-P 算法相对于 CM 算法的改进幅度值为正（表 4-9 中加粗部分），说明本文提出的 IHTS-P 算法相对 CM 算法有大幅改进。针对中小型算例（A 算例集），除了极少数情况下，IHTS-P 算法和 CM 算法的求解效果相当，其他情况相对了 CM 算法都有较大幅度的改进。针对大型算例（B 算例集），IHTS-P 算法的性能优势更加明显，它基于 CM 算法的求解结果，有近 1/2 的算例的改进幅度接近 50%!，这归因于 IHTS-P 算法采用的快速迭代的启发式树搜索算法，加入高效的分支策略，以及带禁忌的扰动机制，使得解能够在有效的时间内不断优化。在求解效率方面，根据附录I中的表格 1， IHTS-P 算法的求解效率明显优于 CM 算法（CM 算法在 3min 内无法得到合法解），这主要归功于我们有效的剪枝策略加速了算法的求解速度。

表 4-10 IHTS-P **算法与** CM **算法结果对比——改进算例占比**

Criteria	Set	%		
		<	=	>
Avg.	A	90.00	10.00	0.00
	B	100.00	0.00	0.00

表 4-10 同样给出了 CM 和 IHTS-P 算法的计算结果对比，比较的是 IHTS-P 算法相较于 CM 算法改进算例的数量相对于测试算例总量的占比。表中列（<）、（=）和（>）对应的含义为 IHTS-P 算法相对于 CM 算法有改进、IHTS-P 算法与 CM 算法的求解结果相当和 IHTS-P 算法的求解结果比 CM 算法差。针对 A 算例集，IHTS-P

算法改进了 90%的算例，剩余的 10%算例是与 CM 算法的求解结果一致；而 B 算例集中这种结果更甚，IHTS-P 算法相对于 CM 算法改进了 B 算例集中的所有算例！由此看来，IHTS-P 算法相对于 CM 算法的求解结果、求解效率和算法的稳定性上都极大的优势。

4.4.3 IHTS-P 算法与 IM 算法对比

表 4-11 IHTS-P 算法与 IM 算法对比——改进幅度

A Set				B Set			
ID	IN	Avg.(%imp)		ID	IN	Avg. (%imp)	
		3min	60min			3min	60min
A1	5	0.00	0.00	B1	68	**+31.36**	**+36.06**
A20	17	**+61.78**	**+8.02**	B20	124	**+41.65**	**+38.00**
A17	21	**+46.84**	**+11.47**	B6	204	**+48.78**	**+32.74**
A6	37	**+38.64**	**+17.28**	B5	207	**+6.65**	0.00
A16	38	**+51.28**	**+35.02**	B10	214	**+38.22**	**+25.10**
A19	47	**+40.96**	**+34.57**	B7	241	**+45.53**	**+24.27**
A12	50	**+50.65**	**+31.20**	B9	247	**+52.38**	**+40.20**
A7	57	**+42.57**	**+43.23**	B17	247	**+46.27**	**+18.76**
A9	63	**+44.44**	**+39.19**	B18	258	**+47.86**	**+30.51**
A3	68	**+45.90**	**+32.90**	B4	261	**+54.64**	**+38.86**
A4	68	**+41.14**	**+21.08**	B14	267	**+45.02**	**+39.92**
A2	72	**+53.51**	0.00	B11	274	**+41.89**	**+42.49**
A18	73	**+47.47**	**+22.44**	B16	300	**+50.11**	**+39.95**
A10	86	**+43.17**	**+35.76**	B3	332	**+48.09**	**+26.25**
A11	86	**+47.33**	**+43.02**	B8	334	**+49.17**	**+37.45**
A5	97	**+49.07**	**+44.16**	B19	371	**+46.18**	**+37.61**
A8	129	**+45.68**	**+48.22**	B2	383	**+48.57**	**+30.65**
A13	272	**+49.44**	**+39.79**	B15	431	**+51.27**	**+40.57**
A14	361	**+54.73**	**+39.82**	B12	439	**+47.77**	**+28.18**
A15	392	**+53.05**	**+35.72**	B13	656	**+62.30**	**+43.25**

根据 4.4.1 和 4.4.2 小节得出的结论，IM 算法相对于 CM 算法性能更优和 IHTS-P 算法相对于 CM 算法性能优势较大。为了分析 IHTS-P 算法的求解质量，本节将比较 IHTS-P 算法和 IM 算法在不同规模不同类型的算例集上的求解结果的差异，分析

两种算法各自的优势和劣势。我们分别选取算例集 A、 B 中共 35 组算例，每种算例我们选取单次运行时间为 3min 下 10 次计算中平均（Avg.）的目标函数值的改进幅度、改进算例占比和运行时间为 60min 下统计 5 次计算中平均（Avg.）的目标函数值的改进幅度、改进算例占比。这主要考虑到运行时间越久，迭代次数越多，算法在求解结果上的随机性就会越低。

如表 4-11 所示，在给定的几乎测试算例测试的结果下，IHTS-P 算法相较于 IM 算法在 3min 和 60min 时间内的改进结果幅度为正值，即本文提出的 IHTS-P 算法相对于模型中的 IM 算法有改进。对于给定的算例，无论在 3min 的运行时间内、还是 60min 的运行时间内，IHTS-P 算法相对于 IM 算法几乎都能获得更好的结果；对于极个别算例（A1、A2、B5）在 3min 或 60min 内，IHTS-P 算法和 IM 算法的改进效果相当。这说明 IHTS-P 算法在不同的求解时间内都能得到较好结果。

表 4-12 也给出了 IHTS-P 和 IM 算法的计算结果对比，但比较的是 IHTS-P 算法相较于 IM 算法改进算例的数量相对于测试算例总量的占比。我们分别统计了算例集 A、B 中的情况如表所示，列(<)表明 IHTS-P 算法计算结果相对于 IM 算法更小，即 IHTS-P 算法相对于 IM 算法计算结果更有优势；列(=)表明 IHTS-P 算法计算结果和 IM 算法计算结果一致，即两种算法的优势相当；列(>)表明 IHTS-P 算法计算结果相对于 IM 算法更大，即 IM 算法相对于 IHTS-P 算法计算结果更有优势。从表中可以明显看出 3min 运行时间内在 A 算例集中，95%的算例在 IHTS-P 算法的求解效果更好，而在更大规模的 B 算例集中，IHTS-P 算法的求解性能已经完全超过了 IM 算法，它所测试的算例的结果都要比 IM 算法更加优异；60min 的运行时间内 IHTS-P 算法相对于 IM 算法改进算例的占比也同样具有压倒性的优势（A 算例集中 90%改进，B 算例集中 95%改进）；而无论是在 3min 还是 60min 内的运行时间内都不存在 IHTS-P 算法相对于 IM 算法改进值为负的情况。这归功我们设计的启发式树搜索算法，通过迭代优化和禁忌扰动机制来对于解进行不断优化，而 IM 算法属于混合整数规划算法范畴需要借助商用求解器，现有的求解效果最好的求解器是 gurobi，但是该类求解器在给定时间内求解大规模复杂性算例时并无太大优势（本文中的算例都源自于实际应用）。

表 4-12 IHTS-P 算法与 IM 模型对比——改进算例占比（3min）

Criteria	Set			
		<	=	>
3min	A	95.00	5.00	0.00
	B	100.00	0.00	0.00
60min	A	90.00	10.00	0.00
	B	95.00	5.00	0.00

4.4.4 与当前最优解对比

表 4-13 模型与 IHTS-P 算法计算结果与最优解对比

Set	Algorithm	ID = 1	ID = 20	ID = 17	ID = 6	ID = 16
		IN = 5	IN= 17	IN= 21	IN=37	IN= 58
		$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$
A	CM	0.00	119.62	0.00	130.45	72.13
	IM	0.00	90.97	12.96	38.72	80.22
	IHTS-P	0.00	75.66	0.00	14.75	17.11
Set	Algorithm	ID = 19	ID = 12	ID = 7	ID = 9	ID = 3
		IN =47	IN= 50	IN= 57	IN= 63	IN= 68
		$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$
A	CM	61.19	146.04	176.71	70.02	97.73
	IM	107.06	54.35	70.52	51.24	42.55
	IHTS-P	35.48	6.20	**-3.19**	**-8.04**	**-4.35**
Set	Algorithm	ID = 4	ID = 2	ID = 18	ID = 10	ID = 11
		IN = 68	IN= 72	IN= 73	IN= 86	IN= 86
		$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$
A	CM	11.99	139.74	91.82	64.18	52.76
	IM	3.35	77.69	21.14	28.47	54.02
	IHTS-P	**-18.44**	77.69	**-6.05**	**-17.47**	**-12.24**
Set	Algorithm	ID = 5	ID = 8	ID = 13	ID = 14	ID = 15
		IN = 97	IN= 129	IN= 272	IN= 361	IN= 392
		$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$	$\%dev.$
A	CM	88.96	50.89	49.63	108.70	80.20
	IM	63.34	61.53	27.33	68.52	73.17
	IHTS-P	**-8.78**	**-16.35**	**-23.34**	1.41	11.31

$$\%dev = (Obj - Obj_{best})/Obj_{best} * 100 \quad (4\text{-}3)$$

给定的算例集 A(A1~A20)中存在当前最优解决方案，为了比较本文中提出的各个算法的求解结果和最优解[2]的差距，本文给出了 CM、IM 和 IHTS-P 算法在算例集 A 中求得的结果相对于最优解的偏差（$\%dev.$），给定算例集 A 中的 20 组算例，每种算例依旧是统计 5 次运行时间为 60min 中计算结果的平均值，如表 4-13 所示。

如下表所示，其中列（$\%dev.$）表示的是本文提出的各算法与当前最优解的差距，具体的求解方式如公式(4-3)所示，其中 Obj 表示当前算法的目标函数值，Obj_{best} 表示当前最优解的对应的目标函数值。可以看出在绝大多数情况下，CM 算法的求解结果相对于当最优解的差距最大；IM 算法次之；IHTS-P 算法的求解结果与当最优解的差距最小，绝大多数情况不超多 20%，甚至有接近 50%的算例在该算法下的求解结果相对于当前最优解有改进。显然无论是与当前最优解的相差幅度还是改进算例的比例，IHTS-P 算法都是最优的。

4.4.5 不同的运行时间求解效果对比

在实际应用场景中，算法的求解时间往往的有限的，不能无限制的让一个算法一直运行下去，因此算法在固定时间内的求解效果也是评估算法性能的一个重要参考。

表 4-14 列出了在 3min 和 60min 内，本文中提出的三种算法：CM、IM、IHTS-P 算法在算例集 A、B（共 40 组算例）中求得可行解的比例，每种算例依旧是统计 10 次取最优的情况（若一次或超过一次求得可行解，则认为该算法可求解出该算例）。如表所示，IM 算法和 IHTS-P 算法不论在 3min 还是 60min 都能得到所有算例的可行解，CM 算法在 60min 的运行时间下可以求解出所有算例，但是它在 3min 内无法求解任何算例，这和我们在后文附录I表格 1 中 3min 下各算法的求解结果中 CM 算法的结果均为 timeout 一致。这主要是因为 CM 算法采用的完整模型思想，庞大的约束和决策变量在计算之前需要全部加载到模型中，而这个过程需要消耗大量的时间，因此在较短的运行时间内（3min）CM 毫无优势可言，但是当给定充足的时间后该算法还是能得到各算例的可行解。因此 IM 和 IHTS-P 算法相对于 CM 算法的资源占用更

[2] http://www.roadef.org/challenge/2018/en/sprintresult.php

少、求解效率更高。

表 4-14 各算法在 3min 和 60min 求解效果对比——可行解占比

A Set			B Set		
Algorithm	%		Algorithm	%	
	3min	60min		3min	60min
CM	0.00	100.00	CM	0.00	100.00
IM	100.00	100.00	IM	100.00	100.00
IHTS-P	100.00	100.00	IHTS-P	100.00	100.00

为了更详细比较每一组算例下各算法的命中率（每组算例得到可行解的概率），我们在附录I中表格 3 统计了 3min 和 60min 下各算法求解算例集 A、B 中每一组算例 10 次得到可行解的概率，其中算例按照规模大小升序排列。如附录I中表格 3 所示，可以更详细的看出在 3min 的运行时间内，IHTS-P 算法在单个算例中的命中率最高、IM 算法次之，CM 算法最差。具体地，CM 算法相对于 IM 算法和 IHTS-P 算法没有任何竞争力，它无法求得任何算例的可行解；IM 算法除了占总算例 17.5%的较大规模算例无法每次得到可行解，其他中小规模都有 100%的命中率；IHTS-P 算法几乎每次运行都行得到算例的可行解，除了个别大规模算例（B12）只能达到 90%的命中率。而在 60min 的运行时间内，IHTS-P 算法和 IM 算法在单个算例中的命中率相当且最高，CM 算法次之。其中 CM 算法相对于 3min 内运行结果的改进幅度最大，它除了少部分（12.5%）大规模算例得不到可行解外，其他情况下均能 100%命中可行解；通过增大运行时间，IM 算法针对各算例的命中率也都提高了且均达到了 100%的命中率；IHTS-P 算法依然保持 3min 时间内求解的高命中率，在 60min 时间内下它针对所有的算例均能达到 100%的命中率！可见单个算例的命中率和运行时间成正相关，并且我们提出的 IHTS-P 算法相对于前两种混合整数规划算法 CM 和 IM 算法的求解效率更高，算法的稳定性更强。这主要归因为我们提出的 IHTS-P 算法中启发式树搜索部分（3.4 节）采用了深度优先搜索算法并返回单个 $1-cut$ 的精确最优解来保证算法的稳定性。

4.5 本章小结

本章主要分析了求解二维矩形切割优化问题的模型算法和 IHTS-P 算法的性能。首先描述了测试算例的来源和结构，并交代了测试环境和介绍了输出文件的含义以及对应的可视化展示；然后基于统计学的相关方法对测试算例集进行分析，来找到算法实现的突破口；接着给定了模型和算法涉及到的全局参数数值和相关说明；最后分析比较本文中提出的模型、IHTS-P 算法的求解结果，再与最优解进行比较，比较本文中提出的算法在不同运行时间上的求解效果，通过实验结果分析，本文中提出的 IHTS-P 算法相较于模型算法无论是求解效果还是求解速率都具有一定的优势。

5 总结与展望

5.1 总结

本文求解的二维矩形切割优化问题属于组合优化问题的范畴，它源自于工业生产和生活中常常涉及到的原料或半成品的切割工序。在工业应用方面，研究切割优化问题对于降低原料消耗、提升企业经济效益、推动可持续发展都有重要的经济和环保价值；在学术研究方面，由于切割优化问题属于 NP 难问题，解决该类问题甚至于推动其他相关问题的研究都有着极大的理论意义。因此，研究切割优化问题是关乎工业界实际应用和学术界理论研究的重要举措，吸引了国内外众多学者参与其中，深入研究。

二维矩形切割优化问题在工业界应用的最为广泛，有十分诱人的发展前景。而本文研究的二维矩形切割优化问题源于真实的工业问题，相比于传统的同类问题来说问题规模更加庞大，目标函数和实际约束更加复杂，抽象成可研究的问题的难度更大。在本文中我们首先对问题的来源进行了简单的介绍，对于涉及到的专业术语和行业概念进行了简单的科普，对和问题相关的数据进行了说明、对相应的约束和目标进行了归纳总结。然后，我们基于实际问题进行抽象和建立数学模型，包括数据、目标、约束、决策，我们分别实现了两种数学模型：完整模型和迭代模型，其中完整模型是直接求解问题、而迭代模型则是将原问题拆分成子问题迭代求解的。接着，我们在本文提出了求解二维矩形切割优化问题的启发式树搜索算法，该算法的思想源自于经典的分支定界算法，由于该问题的解（切割模式）比较复杂，我们在文中简单介绍了解的深度优先树的表示方法，在具体求解过程中：通过深度优先搜索指导原料的切割顺序，加入分支策略对解空间进行细分，而剪枝策略则是对没有搜索意义的解空间进行剪枝便于算法加速，另外为了提高求解质量和求解效率，本文创新性地引入了迭代优化策略和禁忌扰动机制：通过迭代优化策略来反复引导启发式树搜索向更有潜力的解空间搜索，通过禁忌扰动机制来增强解的疏散性。

由于本文中的实验样例都是源自于真实的应用场景，数据复杂。因此我们对于算法中涉及到的相关参数进行了简要说明；对于算例的构成运用适当的统计分析方法

进行分析，便于提取有用信息；对于测试的结果我们设计了比较完备的检查和展示方案；本文研究的问题源自于实际问题，暂无合适的启发式算法和 IHTS-P 算法进行比较，因此最后我们只对本文中提出的 IHTS-P 算法分别与混合整数规划算法：CM 和 IM 算法在专业的算例集上进行了详细的比较：改进幅度的比较，相对于最优解的比较和不同的运行时间下的求解效果比较，分析得出我们设计的 IHTS-P 算法相对于混合整数规划算法具有显著的性能优势。

5.2 展望

本文中提出的 IHTS-P 算法，针对一些大规模算例或结构更为复杂的算例，求解开销比较大，解的质量仍有提升空间。后续将会在以下几个方面做进一步的研究和优化：

- 与当前最前沿的算法进行性能对比实验比较。
- 改进当前的混合整数规划算法，考虑将整数规划算法与启发式算法相结合，利用各自算法的优势来改进解空间。
- 采用一种多层次嵌套的启发式树搜索算法来提升 IHTS-P 算法的求解性能，将整个问题进行分割，在全局范围内采用类似于 A*的算法快速构造解空间，在局部范围利用 IHTS-P 算法来保证解的优度。
- 探索其他算法来解决此问题，运用同为混合整数规划中的经典算法：列生成算法的思想来解决本问题，将问题划分为主问题和子问题分别求解。

致谢

论文行笔至此，发觉不知不觉我已在华科呆了七个年头了。醉晚亭的荷花开了又谢了，三四月份的梧桐雨来了又走。三年前，我在华科送走了一批又一批同学，当时离别的情绪还没有那么浓烈，而今也该到我说再见的时候了。

华科，一座古老而又年轻的校园，装载着我太多的回忆。它明德厚学的治学态度、开放包容的校园文化、睿智渊博的老师和热情好学的同学们都深深影响着我，让我的内心更加阳光、纯粹和坚定。我希望在人生漫漫长路上能一直保有初心，坚持美好。

感谢我的父母给我提供了精神上和物质上的良好环境，这些年无怨无悔地支持我的学业和选择。我因离家甚远，不能常伴膝下，每每只能通讯寄托牵挂，托清风捎去安康，愿时光慢些走吧。

感谢再一次指导我研究生论文的吕志鹏教授。在我的研究生期间，吕老师渊博的学识和严谨的治学态度给了我许多建设性的意见和启发性的思考。研一期间，我曾有过短暂的出国考虑，吕老师理解包容我的决定，以我的个人发展为首，认真和我分析利弊，后来因为种种原因搁置，吕老师依然耐心指导我完成了货物转运、波长路由分配等相关项目，师恩永难忘。

感谢实验室的兄弟姐妹们，我完成这篇论文离不开张佳璐学弟和苏宙行师兄无私的指导和帮助。感谢在华科的老友胡钰湄等，在撰写论文的过程中我们互相鼓励，劳逸结合，一直保持愉快的心情。特别感谢一路相伴的罗茂同学，这些年你既是我的师兄又是我的朋友，我前进的路途离不开你始终如一的鼓励和相伴，你是我的sunshine。

最后由衷地感谢各位专家老师们在论文答辩和审阅过程中的建议和指导。

随着论文答辩的结束，我的学生生涯也暂时画上句号，感谢一路上给予我帮助的所有人，愿我们一路顺遂美好。

参考文献

[1] Bellmore M, Nemhauser G L. The traveling salesman problem: a survey. Operations Research, 1968, 16(3): 538-558

[2] Dantzig G B, Ramser J H. The truck dispatching problem. Management Science, 1959, 6(1): 80-91

[3] Lodi A, Martello S, Monaci M. Two-dimensional packing problems: A survey. European Journal of Operational Research, 2002, 141(2): 241-252

[4] Tovey C A. A simplified NP-complete satisfiability problem. Discrete Applied Mathematics, 1984, 8(1): 85-89

[5] Kantorovich L V. Mathematical methods of organizing and planning production. Management Science, 1960, 6(4): 366-422

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

[7] Balas E, Zemel E. An algorithm for large zero-one knapsack problems. Operations Research, 1980, 28(5): 1130-1154

[8] Pardalos P M, Xue J. The maximum clique problem. Journal of Global Optimization, 1994, 4(3): 301-328

[9] Lawler E L, Wood D E. Branch-and-bound methods: A survey. Operations Research, 1966, 14(4): 699-719

[10] Stuart T E, Podolny J M. Local search and the evolution of technological capabilities. Strategic Management Journal, 1996, 17(S1): 21-38

[11] Dunstan F, Welsh D. A greedy algorithm for solving a certain class of linear programmes. Mathematical Programming, 1973, 5(1): 338-353

[12] Hertz A, de Werra D. Using tabu search techniques for graph coloring. Computing, 1987, 39(4): 345-351

[13] Dorigo M, Blum C. Ant colony optimization theory: A survey. Theoretical Computer Science, 2005, 344(2-3): 243-278

[14] Johnson E L, Nemhauser G L, Savelsbergh M W. Progress in linear programming-based algorithms for integer programming: An exposition. INFORMS Journal on

Computing, 2000, 12(1): 2-23

[15] Bellman R. Dynamic programming. Science, 1966, 153(3731): 34-37

[16] Davis L S. Shape matching using relaxation techniques. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1979 (1): 60-72

[17] Martello S, Vigo D. Exact solution of the two-dimensional finite bin packing problem. Management Science, 1998, 44(3): 388-399

[18] Raffensperger J F. The generalized assortment and best cutting stock length problems. International Transactions in Operational Research, 2010, 17(1): 35-49

[19] Zak E J. Row and column generation technique for a multistage cutting stock problem. Computers & Operations Research, 2002, 29(9): 1143-1156

[20] Marchand H, Martin A, Weismantel R, et al. Cutting planes in integer and mixed integer programming. Discrete Applied Mathematics, 2002, 123(1-3): 397-446

[21] Yanasse H H, Morabito R. Linear models for 1-group two-dimensional guillotine cutting problems. International Journal of Production Research, 2006, 44(17): 3471-3491

[22] Silva E, Alvelos F, de Carvalho J V. An integer programming model for two-and three-stage two-dimensional cutting stock problems. European Journal of Operational Research, 2010, 205(3): 699-708

[23] Dyckhoff H. A new linear programming approach to the cutting stock problem. Operations Research, 1981, 29(6): 1092-1104

[24] Furini F, Malaguti E, Thomopulos D. Modeling two-dimensional guillotine cutting problems via integer programming. INFORMS Journal on Computing, 2016, 28(4): 736-751

[25] Russo M, Sforza A, Sterle C. An exact dynamic programming algorithm for large-scale unconstrained two-dimensional guillotine cutting problems. Computers & Operations Research, 2014, 50:97-114

[26] Hifi M, M'Hallah R, Saadi T. Approximate and exact algorithms for the double-constrained two-dimensional guillotine cutting stock problem. Computational Optimization and Applications, 2009, 42(2): 303-326

[27] Bekrar A, Kacem I. An exact method for the 2D guillotine strip packing problem. Advances in Operations Research, 2009, 2009:1-20

[28] Na B, Ahmed S, Nemhauser G, et al. A cutting and scheduling problem in float glass manufacturing. Journal of Scheduling, 2014, 17(1): 95-107

[29] Alvarez-Valdes R, Parajon A, Tamarit J M. A computational study of LP-based heuristic algorithms for two-dimensional guillotine cutting stock problems. OR Spectrum, 2002, 24(2): 179-192

[30] Gilmore P, Gomory R E. Multistage cutting stock problems of two and more dimensions. Operations Research, 1965, 13(1): 94-120

[31] Furini F, Malaguti E, Durán R M, et al. A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size. European Journal of Operational Research, 2012, 218(1): 251-260

[32] Almada-Lobo B, Oliveira J F, Carravilla M A. Production planning and scheduling in the glass container industry: A VNS approach. International Journal of Production Economics, 2008, 114(1): 363-375

[33] Dusberger F, Raidl G R. Solving the 3-staged 2-dimensional cutting stock problem by dynamic programming and variable neighborhood search. Electronic Notes in Discrete Mathematics, 2015, 47:133-140

[34] Tiwari S, Chakraborti N. Multi-objective optimization of a two-dimensional cutting problem using genetic algorithms. Journal of Materials Processing Technology, 2006, 173(3): 384-393

[35] Afsharian M, Niknejad A, Wäscher G. A heuristic, dynamic programming-based approach for a two-dimensional cutting problem with defects. OR Spectrum, 2014, 36(4): 971-999

[36] Bortfeldt A, Winter T. A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces. International Transactions in Operational Research, 2009, 16(6): 685-713

[37] Lodi A, Monaci M, Pietrobuoni E. Partial enumeration algorithms for two-dimensional bin packing problem with guillotine constraints. Discrete Applied Mathematics, 2017, 217:40-47

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

[39] Bortfeldt A, Jungmann S. A tree search algorithm for solving the multi-dimensional

strip packing problem with guillotine cutting constraint. Annals of Operations Research, 2012, 196 (1): 53-71

[40] Cui Y, Yang Y, Cheng X, et al. A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem. Computers & Operations Research, 2008, 35(4): 1281-1291

[41] Fleszar K. Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. Computers & Operations Research, 2013, 40(1): 463-474

[42] Lee C Y. An algorithm for path connections and its applications. IRE Transactions on Electronic Computers, 1961, 10(3): 346-365

[43] Tarjan R. Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1972, 1(2): 146-160

[44] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529(7587): 484

[45] 邝孔武, 王晓敏. 信息系统分析与设计. 第四版. 北京:清华大学出版社, 2006. 23-25

[46] 贾俊平. 统计学. 第二版. 北京:清华大学出版社, 2006. 39-43

附录Ⅰ：测试数据

表格 1. 3min 内算例集的求解结果

			Heur					
Instance		**Optimal**	**CM**		**IM**		**IHTS-P**	
ID	**IN**		**Obj**	**Util**	**Obj**	**Util**	**Obj**	**Util**
A1	5	425486	Timeout	Timeout	425486	91.39%	425486	91.39%
A2	72	6222839	Timeout	Timeout	31121350	71.82%	14466761	84.41%
A3	68	3688710	Timeout	Timeout	6983337	85.70%	3777948	91.71%
A4	68	4979130	Timeout	Timeout	8298270	83.48%	4884435	89.54%
A5	97	4824453	Timeout	Timeout	9903315	85.12%	5043375	91.82%
A6	37	4111890	Timeout	Timeout	7690456	84.95%	4718580	90.16%
A7	57	5426010	Timeout	Timeout	14400586	83.00%	8269749	89.48%
A8	129	13662614	Timeout	Timeout	24249778	85.07%	13173089	91.29%
A9	63	4153716	Timeout	Timeout	7704268	85.37%	4280511	91.30%
A10	86	6652381	Timeout	Timeout	12574247	85.00%	7146400	90.88%
A11	86	6869149	Timeout	Timeout	13638747	82.59%	7183408	89.97%
A12	50	2226634	Timeout	Timeout	5131976	85.10%	2532868	92.01%
A13	272	20132943	Timeout	Timeout	38953976	84.58%	19693815	91.55%
A14	361	17788848	Timeout	Timeout	50489118	81.77%	22854549	90.83%
A15	392	18930941	Timeout	Timeout	52678875	81.92%	24731090	90.61%
A16	38	4089743	Timeout	Timeout	9831470	79.39%	4789523	88.63%
A17	21	3617251	Timeout	Timeout	6804781	74.25%	3617251	84.44%
A18	73	6848628	Timeout	Timeout	12895947	82.41%	6773835	89.90%
A19	47	3708944	Timeout	Timeout	8475794	82.95%	5003858	89.13%
A20	17	1467925	Timeout	Timeout	6066892	71.34%	2318575	86.44%
B1	68	-	Timeout	Timeout	9268568	89.29%	6361485	92.38%
B2	383	-	Timeout	Timeout	54160785	85.35%	27852374	91.89%
B3	332	-	Timeout	Timeout	62173978	84.92%	32277215	91.56%
B4	261	-	Timeout	Timeout	34168930	81.27%	15500212	90.53%
B5	207	-	Timeout	Timeout	117434805	73.17%	109627550	74.47%
B6	204	-	Timeout	Timeout	37967727	83.57%	19445599	90.85%
B7	241	-	Timeout	Timeout	34145494	84.62%	18600534	90.99%
B8	334	-	Timeout	Timeout	63374799	84.28%	32216185	91.33%
B9	247	-	Timeout	Timeout	60677802	82.89%	28894201	91.05%
B10	214	-	Timeout	Timeout	67449003	83.69%	41668958	89.25%
B11	274	-	Timeout	Timeout	69163620	82.95%	40190267	89.32%
B12	439	-	Timeout	Timeout	56727677	82.08%	29629570	89.77%

			Heur					
Instance		**Optimal**	**CM**		**IM**		**IHTS-P**	
ID	**IN**		**Obj**	**Util**	**Obj**	**Util**	**Obj**	**Util**
B13	656	-	Timeout	Timeout	133763315	78.37%	50430003	90.57%
B14	267	-	Timeout	Timeout	32188840	84.56%	17696653	90.87%
B15	431	-	Timeout	Timeout	88550111	83.01%	43153970	90.93%
B16	300	-	Timeout	Timeout	52823990	82.27%	26355400	90.29%
B17	247	-	Timeout	Timeout	27852665	84.09%	14964444	90.77%
B18	258	-	Timeout	Timeout	29219221	84.30%	15236140	91.15%
B19	371	-	Timeout	Timeout	46500539	83.84%	25026245	90.60%
B20	124	-	Timeout	Timeout	14758163	84.26%	8611976	90.15%

表格 2. 60min 内算例集的求解结果

			Heur					
Instance		Optimal	CM		IM		IHTS-P	
ID	IN		Obj	Util	Obj	Util	Obj	Util
A1	5	425486	425486	91.39%	425486	91.39%	425486	91.39%
A2	72	6222839	14918729	83.81%	11057099	87.47%	11057099	87.47%
A3	68	3688710	7293540	85.14%	5258400	89.00%	3528210	92.22%
A4	68	4979130	5576190	88.23%	5146050	89.04%	4061070	91.14%
A5	97	4824453	9116223	86.12%	7880373	87.77%	4400733	92.78%
A6	37	4111890	9475800	82.03%	5704050	88.35%	4718580	90.16%
A7	57	5426010	15014280	82.38%	9252330	88.35%	5252670	93.04%
A8	129	13662614	20615474	87.01%	22069604	86.22%	11428454	92.35%
A9	63	4153716	7061976	86.40%	6281946	87.72%	3819876	92.16%
A10	86	6652381	10921681	86.68%	8546281	89.27%	5490361	92.83%
A11	86	6869149	10493239	86.00%	10579909	85.90%	6028129	91.45%
A12	50	2226634	5478364	84.19%	3436804	89.46%	2364664	92.50%
A13	272	20132943	30125673	87.63%	25634883	89.28%	15433503	93.26%
A14	361	17788848	37125888	85.91%	29977218	88.31%	18039228	92.62%
A15	392	18930941	34114241	87.49%	32782091	87.92%	21072011	91.89%
A16	38	4089743	7039733	84.13%	7370363	83.51%	4789523	88.63%
A17	21	3617251	3617251	84.44%	4085911	82.77%	3617251	84.44%
A18	73	6848628	13137018	82.11%	8296338	87.90%	6434538	90.36%
A19	47	3708944	5978414	87.29%	7679714	84.24%	5025044	89.09%
A20	17	1467925	3223795	82.02%	2803285	83.99%	2578585	85.09%
B1	68	-	8785998	89.77%	7068648	91.60%	4519908	94.46%
B2	383	-	38336555	89.16%	34404305	90.16%	23859455	92.97%
B3	332	-	46066733	88.37%	38943743	89.99%	28719893	92.42%
B4	261	-	26476165	84.84%	20081845	88.07%	12278335	92.35%
B5	207	-	236125625	57.52%	108579485	74.65%	108579485	74.65%
B6	204	-	28407277	87.16%	25014307	88.52%	16825597	91.98%
B7	241	-	25535739	88.03%	20367639	90.21%	15424239	92.41%
B8	334	-	45163399	88.26%	39472069	89.58%	24690019	93.22%
B9	247	-	31736977	90.25%	45925177	86.48%	27461257	91.45%
B10	214	-	45384533	88.40%	50645723	87.23%	37934123	90.12%
B11	274	-	48964160	87.28%	64442780	83.91%	37058270	90.07%
B12	439	-	44562847	85.36%	34830127	88.18%	25013947	91.22%
B13	656	-	Timeout	Timeout	71992215	87.05%	40852005	92.22%
B14	267	-	24985600	87.58%	23663080	88.16%	14216050	92.53%
B15	431	-	Timeout	Timeout	62340461	87.40%	37048871	92.11%
B16	300	-	35493557	87.35%	34280177	87.72%	20586317	92.25%
B17	247	-	24341567	85.80%	14220437	91.18%	11552927	92.72%
B18	258	-	Timeout	Timeout	18338926	89.52%	12743896	92.48%
B19	371	-	Timeout	Timeout	32745332	88.05%	20428562	92.19%
B20	124	-	Timeout	Timeout	11514137	87.25%	7138907	91.69%

表格 3. 各算法在 3min 和 60min 内的求解效果对比——命中率

A Set					B Set			
ID	**Heur**	**Hit.(%$rate$)**			**ID**	**Heur**	**Hit. (%$rate$)**	
		3min	60min				3min	60min
A1	CM	0.00	100.00		B1	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A20	CM	0.00	100.00		B20	CM	0.00	0.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A17	CM	0.00	100.00		B6	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A6	CM	0.00	100.00		B5	CM	0.00	0.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A16	CM	0.00	100.00		B10	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A19	CM	0.00	100.00		B7	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A12	CM	0.00	100.00		B9	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A7	CM	0.00	100.00		B17	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A9	CM	0.00	100.00		B18	CM	0.00	0.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A3	CM	0.00	100.00		B4	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A4	CM	0.00	100.00		B14	CM	0.00	100.00

A Set					B Set			
ID	**Heur**	**Hit.(%$rate$)**			**ID**	**Heur**	**Hit. (%$rate$)**	
		3min	60min				3min	60min
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A2	CM	0.00	100.00		B11	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A18	CM	0.00	100.00		B16	CM	0.00	100.00
	IM	100.00	100.00			IM	80.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A10	CM	0.00	100.00		B3	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A11	CM	0.00	100.00		B8	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A5	CM	0.00	100.00		B19	CM	0.00	0.00
	IM	100.00	100.00			IM	90.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A8	CM	0.00	100.00		B2	CM	0.00	100.00
	IM	100.00	100.00			IM	100.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A13	CM	0.00	100.00		B15	CM	0.00	0.00
	IM	60.00	100.00			IM	80.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00
A14	CM	0.00	100.00		B12	CM	0.00	100.00
	IM	70.00	100.00			IM	50.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	90.00	100.00
A15	CM	0.00	100.00		B13	CM	0.00	100.00
	IM	100.00	100.00			IM	20.00	100.00
	IHTS-P	100.00	100.00			IHTS-P	100.00	100.00