“十三五”国家重点图书出版规划项目

排序与调度丛书（二期）

流水作业调度算法设计与性能分析

白丹宇 任 涛 著

清華大學出版社
北京

内 容 简 介

本书针对若干带有释放时间的流水作业调度模型，设计了分支定界算法对小规模问题进行最优求解，同时应用智能优化算法对中等规模问题进行近似求解，其中将精确算法与智能优化相结合的方法为求解类似问题提供了新思路。

本书可作为系统工程、应用数学、运筹学与控制论、计算机软件与理论、工业工程、管理科学与工程等相关专业的教师、研究生、高年级本科生以及科研人员的参考书。

图书在版编目(CIP)数据

流水作业调度算法设计与性能分析/白丹宇，任涛著. —北京：清华大学出版社，2023.11
(排序与调度丛书. 二期)
ISBN 978-7-302-64964-9

Ⅰ. ①流… Ⅱ. ①白… ②任… Ⅲ. ①流水生产－生产调度－算法设计 Ⅳ. ①F406.2

中国国家版本馆 CIP 数据核字(2023)第 243204 号

责任编辑：陈凯仁
封面设计：常雪影
责任校对：薄军霞
责任印制：沈 露

出版发行：清华大学出版社
网 址：https://www.tup.com.cn，https://www.wqxuetang.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-83470000 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市龙大印装有限公司
经 销：全国新华书店
开 本：170mm×240mm **印张**：20.5 **字 数**：379 千字
版 次：2023 年 11 月第 1 版 **印 次**：2023 年 11 月第1 次印刷
定 价：129.00 元

产品编号：084360-01

《排序与调度丛书》编辑委员会

丛书序言

我知道排序问题是从 20 世纪 50 年代出版的一本名为 *Operations Research*(《运筹学》,可能是 1957 年出版)的书开始的。书中讲到了 S. M. 约翰逊(S. M. Johnson)的同顺序两台机器的排序问题并给出了解法。约翰逊的这一结果给我留下了深刻的印象。第一,这个问题是从实际生活中来的。第二,这个问题有一定的难度,约翰逊给出了完整的解答。第三,这个问题显然包含着许多可能的推广,因此蕴含了广阔的前景。在 1960 年左右,我在《英国运筹学》(季刊)(当时这是一份带有科普性质的刊物)上看到一篇文章,内容谈到三台机器的排序问题,但只涉及四个工件如何排序。这篇文章虽然很简单,但我也从中受到一些启发。我写了一篇讲稿,在中国科学院数学研究所里做了一次通俗报告。之后我就到安徽参加"四清"工作,不意所里将这份报告打印出来并寄了几份给我,我寄了一份给华罗庚教授,他对这方面的研究给予了很大的支持。这是 20 世纪 60 年代前期的事,接下来便开始了"文化大革命",倏忽十年。20 世纪 70 年代初我从"五七"干校回京,发现国外学者在排序问题方面已做了不少工作,并曾在 1966 年开了一次国际排序问题会议,出版了一本论文集 *Theory of Scheduling*(《排序理论》)。我与韩继业教授做了一些工作,也算得上是排序问题在我国的一个开始。想不到在秦裕瑗、林诒勋、唐国春以及许多教授的努力下,跟随着国际的潮流,排序问题的理论和应用在我国得到了如此蓬勃的发展,真是可喜可贺!

众所周知,在计算机如此普及的今天,一门数学分支的发展必须与生产实际相结合,才称得上走上了健康的道路。一种复杂的工具从设计到生产,一项巨大复杂的工程从开始施工到完工后的处理,无不牵涉排序问题。因此,我认为排序理论的发展是没有止境的。我很少看小说,但近来我对一本名叫《约翰·克里斯托夫》的作品很感兴趣。这是罗曼·罗兰写的一本名著,实际上它是以贝多芬为背景的一本传记体小说。这里面提到贝多芬的祖父和父亲都是宫廷乐队指挥,当贝多芬的父亲发现他在音乐方面是个天才的时候,便想将他培养成一名优秀的钢琴师,让他到各地去表演,可以名利双收,所以强迫他勤学苦练。但贝多芬非常反感,他认为这样的作品显示不出人的气质。由于贝多芬有如此的感受,他才能谱出如《英雄交响曲》《第九交响曲》等深具人性的伟大

乐章。我想数学也是一样，只有在人类生产中体现它的威力的时候，才能显示出数学这门学科的光辉，也才能显示出作为一名数学家的骄傲。

任何一门学科，尤其是一门与生产实际有密切联系的学科，在其发展初期那些引发它成长的问题必然是相互分离的，甚至是互不相干的。但只要研究继续向前发展，一些问题便会综合趋于统一，处理问题的方法也会与日俱增、深入细致，可谓根深叶茂，蔚然成林。我们这套丛书已有数册正在撰写之中，主题纷呈，蔚为壮观。相信在不久以后会有不少新的著作出现，使我们的学科呈现一片欣欣向荣、繁花似锦的局面，则是鄙人所厚望于诸君者矣。

越民义
中国科学院数学与系统科学研究院
2019 年 4 月

前　言

2013 年，德国政府提出了“工业 4.0”概念，自此智能制造成为大多数工业化国家的热门话题。智能制造的目标之一就是以最有效的方式利用生产资源。在离散制造业中，流水作业调度为提高生产效率、降低运营成本起到了重要作用。例如，宝马汽车公司沈阳制造工厂拥有目前世界上最先进的流水装配线之一，能够同时生产不同型号的小汽车。先进的制造、信息和优化技术，使得该制造工厂实现了零库存生产。其汽车装配过程是信息技术下典型的流水作业调度模型，具有生产柔性大、定制水平高、鲁棒性强等特点。

一般来说，绝大多数流水作业调度问题都具有 NP-难性质，无法在多项时间内求得最优解，因此，一直以来此类问题都是学术界与工业界共同关注的焦点。在学术研究上，一般是采用精确算法（如分支定界、动态规划等）在限定时间内求得小规模问题的最优解，旨在为评价近似算法性能构造标准测试集；在实际应用中，一般是利用近似算法（如智能优化、调度规则等）快速求得较大规模问题的近似解，旨在保证生产过程的连续性。针对一系列复杂流水作业调度问题，作者所在课题组围绕优化性质、算法设计、性能分析等方面展开深入研究。其中，为了模拟动态调度环境，设定每项任务具有各自的释放时间，并在此基础上研究模型的优化性质，设计相应的算法加速策略。

本书是作者研究团队近年来代表性研究成果的总结，重点讨论如何应用分支定界以及智能优化算法求解流水作业调度问题，同时也对某些基于调度规则的启发式进行了渐近性能分析。全书共分为 7 章，主要内容概述如下：第 1 章为绪论，简要介绍调度问题的描述与求解方法、算法及性能分析方法；第 2 章介绍带有释放时间的流水作业调度问题，其中考虑了非线性目标函数；第 3 章介绍了考虑处理器阻塞的流水作业调度问题，其中研究了与客户满意度相关的目标函数；第 4 章介绍了考虑学习效应的流水作业调度问题，其中证明了基于最短处理时间优先启发式的渐近最优性；第 5 章、第 6 章介绍了双代理流水作业调度问题，其中研究了一类基于优势代理优先启发式的理论性能。第 7 章介绍了考虑学习效应的混合流水作业调度问题。

本书的部分研究成果由作者与清华大学张智海副教授合作完成。感谢国

家自然科学基金-面上项目(61873173)对本书的资助与支持。在本书的撰写过程中,东北大学软件学院博士研究生王心悦,硕士研究生张妍、杨丹丹、张钧桓,大连海事大学博士研究生白校源、杨洁、刘天一,硕士研究生张钰琛、李婉宁、李艳慧做了大量工作;在本书的出版过程中,得到了清华大学出版社编辑的支持与帮助,在此一并表示衷心的感谢!

由于作者水平有限,书中的错误和不妥之处在所难免,恳请广大读者批评指正!

作 者

2023 年 6 月

目　录

第1章 绪 论

排序与调度(scheduling)又称为排程,主要研究如何将有限的资源(resource)实时地分配给任务(task),目的是优化给定的目标函数(objective function)。在实际中,资源和任务会呈现不同的表现形式。例如:资源可能是工厂机床、码头起重机、医院手术室等;任务可能是零件毛坯、集装箱、手术医生等。排序与调度是组合优化中的一个重要分支,最早起源于制造业,现在已逐渐发展成为运筹学、系统科学、控制科学、管理科学和计算机科学等多个领域的交叉学科,广泛应用于工程技术和运营管理的各个领域。

流水作业调度(flowshop scheduling)是排序与调度中一类复杂的多阶段决策过程,具有约束条件复杂、加工环节多、生产连续性强等典型特征,应用背景非常广阔。钢铁冶金、机械制造、港口物流等行业的生产、装配、装卸等环节均可归结为流水作业调度模型。1901 年,美国人 Ransom 首次提出了流水装配线概念,并建立了世界上第一条现代汽车装配线,用于大规模汽车生产,这是第一个流水作业模型。1954 年,约翰逊(Johnson)发表了第一篇关于流水作业调度问题的论文,这也是排序与调度领域的第一篇学术论文。此后,经过近 70 年的发展,流水作业调度已成为运筹与管理领域的研究热点。不过,除了一些特殊情况,绝大多数流水作业调度问题都具有 NP-难性质(即无法在多项式时间内求得最优解),相关的理论与应用研究还亟待丰富。

1.1 调度问题的符号与定义

排序与调度问题通常由处理器、任务和目标函数三个要素构成。任务的数量记作 n,处理器的数量记作 m。在本书中,指标 j 表示任务,指标 i 表示处理器。若任务有多道工序(operation),则采用 (i,j) 表示任务 j 在处理器 i 上执行。通常,有如下与任务 j 相关的参数。

加工时间(processing time) $p_{i,j}$:表示任务 j 在处理器 i 上执行所需的时间。若任务 j 只在一台处理器上加工,则记为 p_j。在确定性调度问题中为非负实数;在随机调度中为非负随机变量。

释放日期(release date) r_j:又称为就绪时间。离线环境中,其表示任务 j

最早可以开始执行的时刻;在线环境中,其表示任务 j 到达系统的时刻。

交付期(due date) d_j:又称为交货期或工期,表示事先与客户约定好的交付期限。任务可在交付日期之后完成,但是会导致相应的惩罚,如信誉的损失等。通常,任务的交付日期要大于其释放时间与处理时间总和。

截止日期(deadline) $\overline{d}_j$:表示绝对不允许延误的交付日期。

权重(weight) w_j:是一个优先因子,表示任务 j 相对于系统内其他任务的重要性,也可以表示将任务保留在系统中而产生的实际费用。

运送时间(delivery time) q_j:表示产品完成生产环节之后,用车辆运送到客户手中的时间。对于带有运输的调度问题,运送时间应该与处理时间规模大致相同。若处理时间远大于运送时间,则简化为调度问题;反之,则简化为车辆运输问题。

目前,调度问题通常使用 Graham 等(1979)提出的"三参数表示法"(standard three-field notation)来描述。该表示法记为 $\alpha|\beta|\gamma$:α 规定处理器环境,一般只包含一项;β 描述加工的特征和约束的细节,可能省略,也可能包含多个项目;γ 表示需要优化的目标函数。

α 域中规定的处理器环境有以下内容。

单机(single machine) 1:在调度系统中只有一台处理器,是最简单的机器环境,如订单调度问题。在实际中,单机调度问题并不多见。针对单机调度所进行的最优性质及近似算法研究,主要是为多机调度问题构造下界(lower bound)、设计算法等提供研究思路。

并行机(parallel machine):系统中所有的处理器具有相同的功能,任务 j 只需要在其中任意一台处理器上执行即可。

同速机(identical machine) P_m:所有的处理器具有相同的执行速度。

恒速机(uniform machine) Q_m:处理器的执行速度不同,但每台处理器的速度都是常数。

变速机(unrelated machine) R_m:处理器的执行速度取决于被执行的任务。

车间作业(shop):又称多工序作业,指每项任务有多道工序,需要经过一系列串行处理阶段执行。在同一时刻,每台处理器只能执行一项任务,而且每项任务只能被一台处理器执行。若省略 m,则表示处理器或处理阶段个数为变量。

流水作业(flow shop) F_m:又称为流水车间,其中每项任务需要经过 m 台处理器执行,而且所有任务都必须遵循相同的加工路径(即必须首先在第一台处理器上执行,然后是第二台处理器,以此类推)。

柔性流水作业(flexible flow shop) FF_m:由一系列处理阶段构成,每个阶

段有多个并行处理器，其中某些阶段可能只有一个处理器，但至少有一个阶段存在两个及以上的并行处理器。所有任务按照相同的顺序经过各处理阶段，某道工序只要在相应处理阶段中的任一处理器上执行即可。柔性流水作业调度是经典的并行机调度与流水作业调度问题的融合与扩展。钢铁冶金、港口物流等行业的生产、装卸等环节都可归结为该调度模型。

自由作业(open shop) O_m：又称为开放车间，其中每项任务需要经过 m 台处理器执行，某些工序的处理时间可以为零。每项任务的加工路径没有限制，不同的任务可以有不同的加工路径。调度者需要同时确定每项任务的加工路径及每台处理器的加工顺序。

柔性开放作业(flexible open shop) FO_m：由一系列处理阶段构成，每个阶段有多个并行处理器，其中某些阶段可能只有一个处理器，但至少有一个阶段存在两个及以上的并行处理器。每项任务经过各阶段的顺序并不固定，某道工序只要在相应处理阶段中的任一处理器上执行即可。柔性开放作业调度是经典的并行机调度与开放作业调度问题的融合与扩展。车辆服务、医疗管理等行业的检测维修、导诊导检等环节都可归结为该调度模型。

异序作业(job shop) J_m：又称为作业车间，其中每项任务按照其预先确定的加工路径在系统中的处理器上执行。若规定所有任务的加工路径完全相同，则异序作业转换为流水作业，即流水作业是异序作业的特殊情况。

β 域中说明的加工约束和特定限制可能包括以下多个选项。

释放时间 r_j：任务有各自不同的释放时间。按照任务释放前对其信息的了解程度，又可分为如下两种调度环境。

离线(off-line)环境：该环境中任务的所有信息(如任务的数量、处理时间及释放时间等)都预先知道，调度者可以据此做出全局决策。

在线(on-line)环境：在该环境中，任务的信息不完备，所有的参数只有在其进入系统之后才知道，调度者不知道后续将会发生的情况。在线环境较为符合动态生产环境中任务实时到达的情况。在线环境又分为两种：①任务按释放时间到达(over time)；②任务按列表逐个到达(over list)。本书中研究的在线环境均指任务按释放时间到达。

如果 β 域中不出现 r_j 或在线，则说明所有任务在零时刻都可以利用，属于离线环境。

中断(preemption) prmp：某任务在执行过程中，若允许被其他任务抢占而中断，并在稍后恢复处理，则称该任务可以中断。根据中断后恢复处理的不同方式，可以分为两种情况：①若中断后该任务的处理时间不受影响，则称为可续性中断；②若中断后该任务的处理必须重新开始，则称为重复性中断。当 prmp

不在 β 域中时，表示不允许中断。

同顺序(permutation) prmu：也称为排列或置换，是流水作业调度的一种排序方式。它要求所有任务按照先入先出(first-in，first-out)规则经过每个处理器，这样各台处理器上任务的执行顺序完全相同。这种调度方式的好处是，在对给定的流水作业调度问题进行编码、解码时，只要考虑第一台处理器上的序列即可，大大节约了计算资源。由于本书所研究的流水作业调度模型中只考虑排列排序，故省略该项。

无等待(no-wait) nwt：是一种作业调度中的实际需求，规定任务不能在相邻两道工序之间等待。换言之，任务一旦开始加工，就必须连续进行，直到所有工序完成为止。例如，在轧钢过程中，加热至轧制温度的钢坯必须直接送至轧钢机进行加工，不允许等待，否则会引起钢坯温度下降，一旦温度低于轧制温度，就需要重新加热，这样会消耗大量的能源。

阻塞(block) block：在作业调度中，若处理相邻两道工序的处理器之间缓存容量已满，则上游处理器上已完工的任务无法进入缓冲区等待，只能继续占用该处理器直至下游处理器空闲为止，此现象称为阻塞。通常，在任务体积较大或工序之间需要运输衔接的情况下会发生阻塞现象。

优先约束(precedence constraint) prec：出现于单机或并行机环境中，在某项任务开始执行之前，另一项或多项任务必须已经完工。有几种优先约束的特殊形式：若每项任务最多有一项先行任务和一项后继任务，则称为链式；若每项任务最多有一项后继任务，则称为入树；若每项任务最多有一项先行任务，则称为出树。当 prec 不在 β 域中时，表示任务没有优先限制。

γ 域中表示需要优化的目标函数。目标函数分为正则函数与非正则函数。本书只考虑正则目标函数，它又分为两类：使最大的费用最小(minimax criteria)及使总的费用和最小(minsum criteria)。对于给定的排序，用 C_j 表示任务 j 的完工时间。常见的目标函数主要有如下几种形式。

最大完工时间(makespan) $C_{\max}$：又称为制造期或时间表长度。对于给定的时间表，所有任务中完工时间的最大值称为最大完工时间，即最后一项任务离开系统的时刻。优化该目标函数能够有效地平衡多机调度问题中机器的负载，旨在降低生产成本，减少能源消耗。在实际调度过程中，该目标函数对于最长处理时间优先(longest processing time first，LPT)规则比较敏感。

加权完工时间和(total weighted completion time) $\sum w_j C_j$：对于给定的时间表，表示所有任务的完工时间(线性加权)总和。优化该目标函数能够有效降低在制品库存，减少库存成本。在实际调度过程中，该目标函数对于(加权)最短处理时间优先([weighted] shortest processing time first，[W]SPT)规则

比较敏感。若所有任务的权重都相等，则称为完工时间和(total completion time)，记为 $\sum C_j$。

完工时间 k 次方和(total k-power completion time，$k\geqslant 2$) $\sum C_j^k$：所有任务完工时间求 k ($k\geqslant 1$)次方后的总和。该目标函数可以作为连接最大完工时间和总完工时间的桥梁。当 $k\to\infty$时，该目标等价于最大完工时间。例如，设 3 个任务的完工时间分别为 1.02、1.01 和 1，令 $k=365$，则有 $1.02^{365}\approx 1377.41$，$1.01^{365}\approx 37.78$ 和 $1^{365}=1$，显然，此时除了最大完工时间其余可以忽略不计。当 $k=1$ 时，该目标等于总完工时间。

最大延迟(maximum lateness) $L_{\max}$：定义为 $L_{\max}=\max\{L_j\}$，其中 $L_j=C_j-d_j$ 是任务 j 的延迟时间。优化该目标函数能够有效降低延误率，提高客户满意度。在实际调度过程中，该目标函数对于最早交付日期优先(earliest due date first，EDD)规则比较敏感。

最大送达时间(maximum delivery-completion time) $Q_{\max}$：任务的完工时间与运送时间之和称为送达时间。对于给定的时间表，所有任务中送达时间的最大值称为最大送达时间。该目标函数考虑了任务的物流环节，能够有效平衡完工时间与运送时间，旨在节约物流成本，提高客户满意度。在实际调度过程中，该目标函数对于最长运送时间优先(longest delivery time first，LDT)规则比较敏感。

最大拖期(maximum tardiness) $T_{\max}$：定义为 $T_{\max}=\max\{T_j\}$，其中 $T_j=\max\{L_j,0\}$是任务 j 的拖期时间。

总加权拖期(total weighted tardiness) $\sum w_jT_j$：表示所有任务拖期时间的加权和。若所有任务的权重都相等，则称为总拖期(total tardiness)，记为 $\sum T_j$。

加权误工任务数(weighted number of tardy tasks) $\sum w_jU_j$：定义为 $\sum w_jU_j=\sum_{j=1}^{n}w_jU_j$，其中 $U_j=\begin{cases}1, & C_j>d_j\\ 0, & C_j\leqslant d_j\end{cases}$。若所有任务的权重都相等，则称为误工任务数(number of tardy tasks)，记为 $\sum U_j$。

1.2　调度问题的求解方法

对于给定的排序与调度问题，首先要从计算复杂性的角度来判断其难易程度。计算复杂性是 20 世纪 70 年代以计算机科学为基础建立起来的理论，从算

法的角度定义了问题的“难”和“易”。在计算复杂性理论中，根据问题是否能采用多项式时间算法最优求解来判定其难易程度。若一个判定问题的任何验证都能在多项式时间内完成，且实例为假时，验证结果总是假，当实例为真时至少有一个验证为真，则称其为非确定性多项式（nondeterministic polynominal，NP）问题。显然，P 问题是 NP 问题的子集，因为存在多项式时间解法的问题，总能在多项式时间内验证它。而 NP-难（NP-hard）问题是指所有的 NP 问题都能归约到它，但是它不一定是一个 NP 问题。根据其难度的不同，又可分为强 NP-难（strong NP-hard）和一般 NP-难（ordinary NP-hard）两类，其中任何强 NP-难问题不存在伪多项式时间算法。

一方面，若某调度问题是 P 问题，即存在多项式时间算法，则主要任务就是尽可能地降低最优算法的时间复杂性；或在算法时间复杂性较高时，采用近似算法求解大规模问题。另一方面，若某调度问题是 NP-难问题，则根据算法求解的时间效率将该问题分成小、中、大三种规模进行处理。

（1）针对小规模问题，可以利用精确算法，如分支定界（branch and bound）、动态规划（dynamic programming）等进行最优求解。

分支定界是一种基于枚举的搜索方法，可用于求解整数线性规划等最优化问题。基本思想是将原问题的可行解集不断地分解为多个子集，直至不可再分为止。分支定界算法中三个极其重要的步骤为分支、定界和剪支。分支是分解可行解集的过程。定界是估计子集目标值的过程，其中松弛某些约束条件求得最优解的估计值称为下界（lower bound），而可行解的目标值则称为上界（upper bound）。剪支是根据上界与下界的大小关系剪掉非（或重复）最优解分支的过程。该算法的性能主要取决于分支规则、剪支策略与下界的有效性。

动态规划是一种求解多阶段决策过程的最优化数学方法，其核心思想是将原问题转化为一系列互相联系的单阶段子问题，然后逐个加以解决。应用该方法的关键在于恰当地选取状态变量和决策变量及定义最优值函数，正确地写出基本递推方程，将全局优化问题转化成一族同类型的子问题。然后，从边界条件开始，逐段递推寻优，在每一个子问题的求解中，均利用了它前面的子问题的最优化结果，最后一个子问题所得的最优解，就是整个问题的最优解。

Benders 分解方法是一种基于松弛和投影映射的数学规划方法。其核心思想是将难以求解的原始问题松弛为仅包含整数变量的易于求解的 Benders 主问题，用包含整数变量和连续变量的目标函数项和约束构建 Benders 子问题。子问题生成 Benders cut 并将其加入主问题使下界逐步逼近最优解。Benders 分解方法的核心难点在于数学建模策略的选择和 Benders cut 的生成方法的设计。好的数学建模策略能够有效减少算法迭代次数，加速问题收敛。Benders

cut 的生成使得主问题解空间逐步收紧，逼近原问题的解空间。通过对问题特征的分析，采取合适的建模策略并设计高效的求解方法，设计高质量有效不等式和 Benders cut，引入有效的求解策略，可以加速问题求解，节约计算资源。

拉格朗日松弛算法是一种基于松弛的数学规划方法。其核心思想是将原问题中复杂的约束以惩罚的形式松弛到目标函数中，通过求解拉格朗日对偶问题更新拉格朗日乘子使松弛问题下界逐步逼近原问题的最优解。拉格朗日松弛算法的关键在于拉格朗日乘子的更新方法与拉格朗日上界构造方法的设计。高质量的拉格朗日乘子能够帮助松弛问题获得更紧的下界，对于缩小对偶间隙、节约计算资源有很大的帮助。上界构造方法通过启发式策略将松弛解转化为原问题的高质量可行解，高效的构造方法能够有效地减少问题的迭代次数。本书拟通过对问题特征的分析，采取合适的建模策略并设计高效的乘子更新方法来加速问题收敛。设计基于问题性质的上界构造启发式算法，生成高质量可行解，缩小对偶间隙。拉格朗日松弛算法也可以与其他数学规划算法相结合，例如：与 Benders 分解方法结合，设计广义 Benders 分解算法求解非线性规划问题。

列生成算法是一种基于重构的数学规划方法。其核心思想是将原问题模型重构为 D-W 分解主问题和列生成子问题，其中列生成子问题用以生成主问题的变量。直接应用列生成算法只能获得原问题的松弛解，将其嵌入分支定界算法，设计分支定界算法，可以获得该问题的精确解。列生成算法设计的关键在于原问题与主问题的建模方法和子问题的求解方法。问题的建模方法直接影响该问题能否应用分支定界算法与如何设计子问题的求解算法。子问题的计算效率直接影响单个节点的计算速度。本书拟通过对原问题与定价子问题结构特征的分析，采取合适的建模策略并设计高效的子问题求解方法，设计高质量有效不等式与对偶稳定性公式，引入有效的求解策略加速问题求解，节约计算资源。

(2) 针对中等规模问题，由于输入规模的增加使得很难在允许的时间内求得最优解，因此研究的重点集中在如何利用智能优化(metaheuristic)算法，如差分进化(differential evolution)、粒子群优化(particle swarm optimization)、蚁群优化(ant colony optimization)等算法，在给定时间内快速求得近优解。

差分进化算法通过群体内个体之间的相互合作与竞争产生的群体智能来指导优化搜索的方向。该算法的基本思想是从某一随机产生的初始群体开始，利用从种群中随机选取的两个个体的差向量作为第三个个体的随机变化源，将差向量加权后按照一定的规则与第三个个体求和而产生变异个体，该操作称为变异(mutation)。然后，变异个体与某个预先决定的目标个体进行参数混合，

生成试验个体，这一过程称为交叉。如果试验个体的适应度值优于目标个体的适应度值，则在下一代中试验个体取代目标个体，否则目标个体仍保存下来，该操作称为选择(selection)。在每一代的进化过程中，每一个体矢量作为目标个体一次，算法通过不断地迭代计算，保留优良个体，淘汰劣质个体，引导搜索过程向全局最优解逼近。

粒子群优化算法模拟了鸟群的捕食行为：鸟群在某个区域里随机搜索食物(只有一块食物)，每只鸟都不知道食物的位置，只知道当前的位置与食物的距离。那么找到食物的最优策略的就是在当前离食物最近那些鸟的周围区域进行搜寻。在该算法中，每个优化问题的解都是搜索空间中的一只鸟，称为“粒子”。所有的粒子都有一个由被优化的函数决定的适应值，每个粒子还有一个速度决定它们飞翔的方向和距离。然后粒子们就追随当前的最优粒子在解空间中搜索。该算法通过初始化(initialization)得到一群随机粒子(随机解)，然后通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个“极值”来更新自己。一个是粒子本身所找到的最优解，这个解叫作个体极值；另一个极值是整个种群目前找到的最优解，这个极值是全局极值。

蚁群优化算法是一种用来寻找优化路径的概率型算法，具有分布计算、信息正反馈和启发式搜索的特征，其灵感来源于蚂蚁在寻找食物过程中发现最短路径的行为。该算法求解优化问题的基本思路为：用蚂蚁的行走路径表示待优化问题的可行解，路径较短的蚂蚁释放的信息素量较多，随着时间的推进，较短的路径上累积的信息素浓度逐渐增高，选择该路径的蚂蚁个数也越来越多。最终，整个蚂蚁群会在正反馈的作用下集中到最佳的路径上，此时对应的便是待优化问题的最优解。

(3) 对于大规模问题或者在线环境，一般是构造基于指派规则的启发式(heuristic)算法在较短时间内求得可行解。启发式算法或称为调度规则是一种基于直观或经验构造的算法，是多项式时间算法，可以快速地求得调度问题的可行解。由于启发式算法构造简单，便于理论分析，所以一般都有性能上的保证。经典的启发式算法主要有：约翰逊规则、CDS 算法、Palmer 算法、Gupta 算法、NEH 算法。

约翰逊规则是 Johnson 在 1954 年针对最简单的两机流水作业调度问题提出一种调度规则。考虑每个任务都要在两台机器(或阶段)上加工的情况。假设每台机器一次只能处理一个任务，每个任务必须由第一台机器处理后，再由第二台机器处理。任务在机器上的处理时间是已知的，并且所有任务在机器上加工的次序相同。对于这类问题，约翰逊规则步骤如下。

步骤 1 把任务按工序加工时间分成两个子集：$\xi_1=\{J_j \mid p_{1,j}<p_{2,j}\}$，

$\xi_2=\{J_j \mid p_{1,j}>p_{2,j}\}$。对于满足 $p_{1,j}=p_{2,j}$ 的任务可以分在两个子集中的任一个。

步骤 2　先将集 ξ_1 中的任务按 $p_{1,j}$ 不减排列(SPT 规则),再将集 ξ_2 中的任务按 $p_{2,j}$ 不增排列(LPT 规则)。

为了便于读者理解约翰逊规则的具体用法,下面结合一个例子来进行说明:

例 1-1　考虑排序问题 $F_2 \parallel C_{\max}$ 其中 $n=5$。

	J_1	J_2	J_3	J_4	J_5
M_1	4	4	30	6	2
M_2	5	1	4	30	3

根据约翰逊规则可得,$\xi_1=\{J_1,J_4,J_5\}$,$\xi_2=\{J_2,J_3\}$,ξ_1 中任务按 $p_{1,j}$ 不减排列:$\{J_5,J_1,J_4\}$,ξ_2 中的任务按 $p_{2,j}$ 不增排列:$\{J_3,J_2\}$,所以最优排序为$\{J_5,J_1,J_4,J_3,J_2\}$,时间表长为 $C_{\max}=47$(图 1-1)。

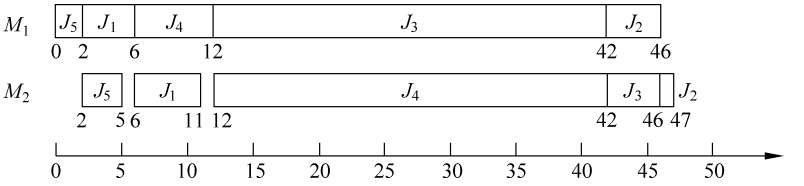

图 1-1　约翰逊规则甘特图

需要说明的是,不满足约翰逊规则的排序也有可能是最优排序,如下面的例 1-2 所示。

例 1-2　考虑排序问题 $F_2 \parallel C_{\max}$ 其中 $n=4$。

	J_1	J_2	J_3	J_4
M_1	4	4	8	1
M_2	5	1	4	20

由约翰逊规则可得最优排序为$\{J_4,J_1,J_3,J_2\}$,$C_{\max}=31$。容易验证,排序$\{J_4,J_3,J_2,J_1\}$也是最优排序,但不满足约翰逊规则。

CDS 算法扩展了约翰逊规则,考虑了流水车间两端机器处理时间对最大完工时间影响较大,适合用于机器规模较大的情况。其步骤如下所示。

步骤 1　假设机器数量为 m,将流水车间第 1 个机器与第 n 个机器组成一个集合,记录为$\{A_1,A_2\}$。

步骤 2　找到集合$\{A_1,A_2\}$中的当前最小处理时间对应的机器 a、任务 b,若机器 a 属于 A_1,则任务 b 先加工;若机器 a 属于 A_2,则任务 b 后加工。若最

小处理时间不唯一，则随机选择一个即可。

步骤 3 将排序后的任务在集合中剔除，重复步骤 2，直到将所有任务排序完成，得到一种序列和对应的最大完工时间 $C_{\max}$。

步骤 4 将机器 2,3,…,$n-1$ 与机器 $n-1$,$n-2$,…,2 分别逐渐加入集合 A_1 和 A_2，集合 A_1 和 A_2 的处理时间为集合内任务处理时间和，更新集合$\{A_1,A_2\}$。重复步骤 2，得到 $n-1$ 个序列与对应最大完工时间。

步骤 5 选择所有序列中最大完工时间值最小的序列作为最终序列。

为了便于阐述 CDS 算法的具体用法，下面结合一个例子来进行说明。

例 1-3 流水问题 $F_7 \parallel C_{\max}$，其中 $n=8$，$m=7$。

	M_1	M_2	M_3	M_4	M_5	M_6	M_7
J_1	13	79	23	71	60	27	2
J_2	31	13	14	94	60	61	57
J_3	17	1	0	23	36	8	86
J_4	19	28	10	4	58	73	40
J_5	94	75	0	58	0	68	46
J_6	8	24	3	32	4	94	89
J_7	10	57	13	1	92	75	29
J_8	80	17	38	40	66	25	88

首先，将第 1 个机器和第 7 个机器组成集合$\{A_1,A_2\}$，见表 1-1。

表 1-1 例 1-3 求解流程(1)

任务编号	A_1	A_2
1	13	2
2	31	57
3	17	86
4	19	40
5	94	46
6	8	89
7	10	29
8	80	88

集合中最小处理时间为 2，属于集合 A_2，所以对应任务 J_1 最后一个加工；除任务 J_1 之外，此时集合中最小处理时间为 8，属于集合 A_1，所以对应任务 J_6 第一个加工；再除去任务 J_6，此时集合中最小处理时间为 10，属于集合 A_1，所

以对应任务 J_7 第二个加工；再除去任务 J_7，此时集合中最小处理时间为 17，属于集合 A_1，所以对应任务 J_3 第三个加工；再除去任务 J_3，此时集合中最小处理时间为 19，属于集合 A_1，所以对应任务 J_4 第四个加工；再除去任务 J_4，此时集合中最小处理时间为 31，属于集合 A_1，所以对应任务 J_2 第五个加工；再除去任务 J_2，此时集合中最小处理时间为 46，属于集合 A_2，所以对应任务 J_5 倒数第二个加工；得到序列$\{J_6,J_7,J_3,J_4,J_2,J_8,J_5,J_1\}$，最大完工时间 $C_{\max}=618$。

然后，将第 1、2 个机器与第 7、8 个机器组成集合$[A_1,A_2]$，见表 1-2。

表 1-2　例 1-3 求解流程(2)

任务编号	A_1	A_2
1	92	29
2	44	118
3	18	94
4	47	113
5	169	114
6	32	183
7	67	104
8	97	113

集合中最小处理时间为 18，属于集合 A_1，所以对应任务 J_3 第一个加工；除去任务 J_3，集合中最小处理时间为 29，属于集合 A_2，所以对应任务 J_1 最后一个加工；除去任务 J_1，此时集合中最小处理时间为 32，属于集合 A_1，所以对应任务 J_6 第二个加工；再除去任务 J_6，此时集合中最小处理时间为 44，属于集合 A_1，所以对应任务 J_2 第三个加工；再除去任务 J_2，此时集合中最小处理时间为 47，属于集合 A_1，所以对应任务 J_4 第四个加工；再除去任务 J_4，此时集合中最小处理时间为 67，属于集合 A_1，所以对应任务 J_7 第五个加工；再除去任务 J_7，此时集合中最小处理时间为 67，属于集合 A_1，所以对应任务 J_8 第六个加工；得到序列$\{J_3,J_6,J_2,J_4,J_7,J_8,J_5,J_1\}$，最大完工时间 $C_{\max}=628$。然后，得到表 1-3 中的信息。

表 1-3　例 1-3 求解流程(3)

序号	A_1	A_2	序列	$C_{\max}$
1	M_1	M_7	$[J_6,J_7,J_3,J_4,J_2,J_8,J_5,J_1]$	618
2	M_1,M_2	M_6,M_7	$[J_3,J_6,J_2,J_4,J_7,J_8,J_5,J_1]$	628

续表

序号	A_1	A_2	序列	$C_{\max}$
3	M_1, M_2, M_3	M_5, M_6, M_7	$[J_3, J_6, J_4, J_2, J_7, J_8, J_5, J_1]$	596
4	M_1, M_2, M_3, M_4	M_4, M_5, M_6, M_7	$[J_3, J_4, J_6, J_7, J_2, J_8, J_5, J_1]$	632
5	M_1, M_2, M_3, M_4, M_5	M_3, M_4, M_5, M_6, M_7	$[J_6, J_3, J_4, J_7, J_2, J_8, J_1, J_5]$	605
6	$M_1, M_2, M_3, M_4, M_5, M_6$	$M_2, M_3, M_4, M_5, M_6, M_7$	$[J_3, J_6, J_4, J_7, J_8, J_2, J_1, J_5]$	584

可知最优序列是$\{J_3, J_6, J_4, J_7, J_8, J_2, J_1, J_5\}$，$C_{\max}=584$(图 1-2)。

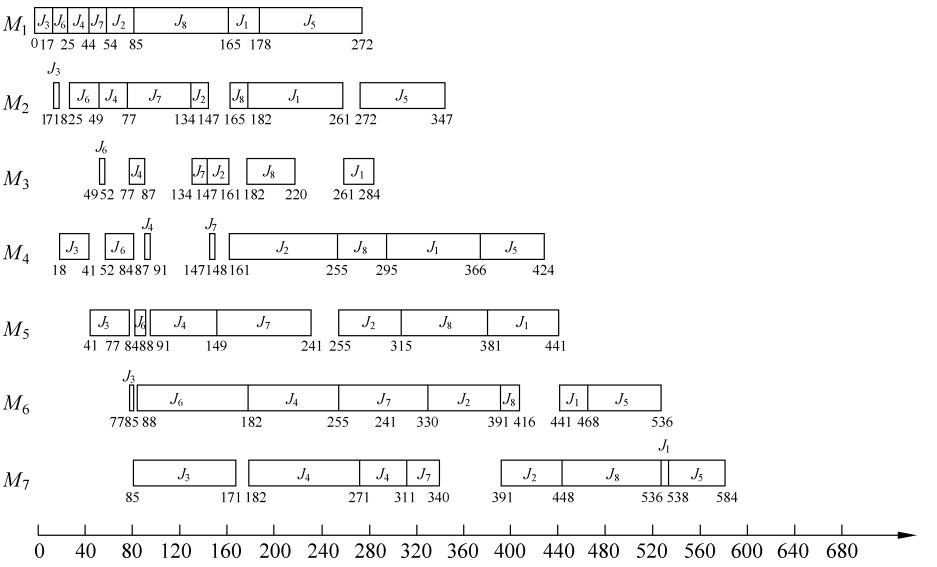

图 1-2 CDS 算法甘特图

Palmer 启发式由 Palmer 于 1965 年提出，该算法根据任务加工时间的斜度顺序指标(slope order index)来调度任务。考虑 n 个任务 m 台机器的情况，每个任务按相同的顺序经过各个机器，目标是在极小化最大完工时间。该算法在排序过程中，按机器的顺序，加工时间趋于增加的任务被赋予较大的优先权数；反之，加工时间趋于减少的任务被赋予较少的优先权数。任务 i 的斜度指标(slope index)$S(i)$定义为

$$S(i)=\sum_{j=1}^{m}\left(j-\frac{m+1}{2}\right)t_{ij},\quad i=1,2,\cdots,n$$

式中，m 为机器数；t_{ij} 为任务 i 在机器 j 上的加工时间。按照 $S(i)$非增的顺序排列任务，获得一个近优调度。为便于阐述算法，给出例 1-4。

例 1-4　考虑流水问题 $F_3 \| C_{\max}$，其中 $n=4, m=3$。

	J_1	J_2	J_3	J_4
M_1	1	2	6	3
M_2	8	4	2	9
M_3	4	5	8	2

首先计算各个任务的斜度指标：

$$S(1)=-1\times1+0\times8+1\times4=3$$
$$S(2)=-1\times2+0\times4+1\times5=3$$
$$S(3)=-1\times6+0\times2+1\times8=2$$
$$S(4)=-1\times3+0\times9+1\times2=-1$$

然后根据 $S(i)$非增的次序排列任务，可得序列$\{J_1, J_2, J_3, J_4\}$与序列$\{J_2, J_1, J_3, J_4\}$，易证，这两个序列均为最优调度。

Gupta 算法于 1972 年提出，用于快速求解流水车间最大完工时间问题。Gupta 考虑到最大完工时间与机器空闲时间和机器间的相互作用密切相关，提出 MINIT 算法、MICOT 算法和 MINIMAX 算法。

MINIT 算法利用空闲时间对每台机器的完工时间分别最小化，步骤如下。

步骤 1　在 n 个任务中选择两个任务(a, b)进行排序，计算所有机器 $M(1, 2, \cdots, m)$的机器空闲时间 $E(ab, M)$，并找到机器 m 的最小空闲时间 $E(ab, m)$。

步骤 2　若 $E(ab, m)$的值唯一，则对应的 ab 作为确定的初始顺序 σ；若 $E(ab, m)$的值不唯一，则选择机器 $m-1, m-2, \cdots, 1$ 上的最小的空闲时间，选择最小值对应的 ab 作为确定的初始顺序 σ。若所有机器的空闲时间都相同，则选择机器 $m, m-1, \cdots, 1$ 上完工时间最大值对应的 ab 作为确定的初始顺序 σ。

步骤 3　在未确定顺序的任务中，检查每个任务，计算所有机器的 $E(\sigma c, m)$。

步骤 4　若 $E(\sigma c, m)$值唯一，则将任务 c 加入确定的初始顺序，更新 abc 为确定的初始顺序 σ；若 $E(\sigma c, m)$值不唯一，选择机器 $m-1, m-2, \cdots, 1$ 空闲时间最少的 c 加入确定的初始顺序，更新 abc 为确定的初始顺序 σ。

步骤 5　循环步骤 4，若剩余两个任务则停止循环，比较两种不同的排序，选择最大完成时间较小者作为最终顺序。

MICOT 算法与 MINIT 算法相似，只是 MICOT 算法不使用空闲时间，直接对每台机器的完成时间分别最小化，步骤如下。

步骤 1　在 n 个任务中选择两个任务(a, b)进行排序，计算所有机器 $M(1, 2, \cdots, m)$的机器完成时间 $C(ab, M)$，并找到机器 m 的最小完成时间$C(ab, m)$。

步骤 2 若 $C(ab,m)$的值唯一，则对应的 ab 作为确定的初始顺序 σ；若 $C(ab,m)$的值不唯一，则选择机器 $m-1,m-2,\cdots,1$ 上的最小的完成时间，选择最小值对应的 ab 作为确定的初始顺序 σ。若所有机器的完成时间都相同，则选择机器 $m,m-1,\cdots,1$ 上完工时间最大值对应的 ab 作为确定的初始顺序 σ。

步骤 3 在未确定顺序的任务中，检查每个任务，计算所有机器的 $C(\sigma c,m)$。

步骤 4 若 $C(\sigma c,m)$值唯一，则将任务 c 加入确定的初始顺序，更新 abc 为确定的初始顺序 σ；若 $C(\sigma c,m)$值不唯一，选择机器 $m-1,m-2,\cdots,1$ 完成时间最少的 c 加入确定的初始顺序，更新 abc 为确定的初始顺序 σ。

步骤 5 循环步骤 4，当剩余两个任务则停止循环，比较两种不同的排序，选择最大完成时间较小者作为最终顺序。

MINIMAX 算法是 Gupta 提出的启发式算法中最简单的一种，它模仿约翰逊规则在解决两机和特殊三机问题的概念，从两端建立时间表，将处理时间短的任务安排在前列，将处理时间长的任务安排在后列，具体步骤如下。

步骤 1 设 k 和 k'分别表示前列和后列安排任务的数量，初始阶段令 $k=k'=0$。

步骤 2 找到在所有机器上处理时间最短的任务 a。若最短工作时间对应的任务不止一个，则选择在机器 $m,m-1,\cdots,1$ 具有最大处理时间的任务，记为 a，$k:=k+1$。

步骤 3 若 $k+k'=n$，则进行步骤 6，否则进行步骤 4。

步骤 4 找到在所有机器上处理时间最长的任务 b。若最长处理时间对应的任务不止一个，则选择在机器 $m,m-1,\cdots,1$ 具有最小处理时间的任务，记为 b，$k':=k'+1$。

步骤 5 若 $k+k'=n$，则进行步骤 6，否则返回步骤 2。

步骤 6 计算最大完工时间，得到最优顺序。

为了便于阐述 MINIT 算法、MICOT 算法和 MINIMAX 算法的具体用法，下面结合一个例子来进行说明。

例 1-5 考虑排序问题 $F_m \| C_{\max}$，其中 $m=3,n=6$，分别用 MINIT 算法、MICOT 算法和 MINIMAX 算法求解。

	J_1	J_2	J_3	J_4	J_5	J_6
M_1	5	6	30	2	3	4
M_2	8	30	4	5	10	1
M_3	20	6	5	3	4	4

(1)MINIT 算法

首先任选两个任务排序，计算每个机器的空闲时间(图 1-3)，$E(12,2)=5$，$E(12,3)=13+(43-33)=23$。

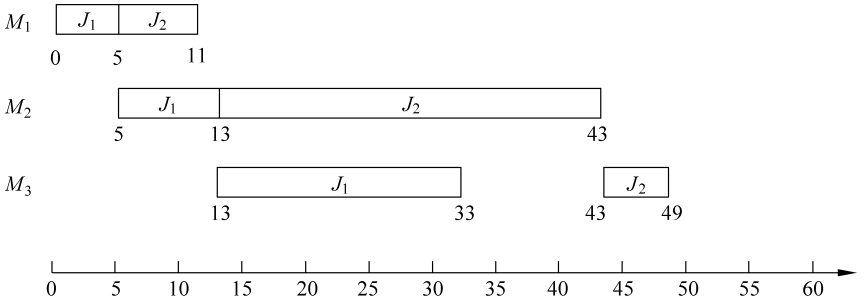

图 1-3　MINIT 算法机器空闲图

依次求得所有可能排序的 $E(ab,m)$，如表 1-4 所示。

表 1-4　例 1-5 求解流程(1)

任务对	$E(ab,m)$		$C(ab,m)$		
(a,b)	2	3	1	2	3
(1,2)	5	23	11	42	49
(1,3)	27	19	35	39	44
(1,4)	5	13	7	18	36
(1,5)	5	13	8	23	37
(1,6)	5	13	9	14	37
(2,1)	6	38	11	44	64
(2,3)	6	36	36	40	47
(2,4)	6	36	8	41	45
(2,5)	6	40	9	46	50
(2,6)	6	36	1	37	46
(3,1)	31	38	35	43	63
(3,2)	32	61	36	66	72
(3,4)	30	34	32	39	42
(3,5)	30	39	33	44	48
(3,6)	30	34	34	35	43
(4,1)	2	12	7	15	35
(4,2)	3	35	8	38	44
(4,3)	27	33	32	36	41
(4,5)	2	14	5	17	21
(4,6)	2	7	6	8	14

续表

任务对	$E(ab,m)$		$C(ab,m)$		
(5,1)	3	17	8	21	41
(5,2)	3	39	9	43	49
(5,3)	23	33	33	37	42
(5,4)	3	14	5	18	21
(5,6)	3	13	6	14	21
(6,1)	8	13	9	17	37
(6,2)	9	36	10	40	46
(6,3)	33	34	34	38	43
(6,4)	5	7	6	11	14
(6,5)	6	13	7	17	21

由于 $\min E(ab,3)=E(46,3)=E(64,3)$，且 $E(46,2)<E(64,2)$，所以(4,6)为初始顺序 σ，继续计算 $E(\sigma c,m)$，按步骤继续排序，如表 1-5 所示。

表 1-5 例 1-5 求解流程(2)

σ	a	$E(\sigma a,m)$		确定的顺序
		$m=2$	$m=3$	
46	1	5	12	$\sigma=465$
	2	6	35	
	3	30	33	
	5	3	12	
465	1	3	16	$\sigma=4651$
	2	3	38	
	3	23	32	

排序$\{J_4,J_6,J_5,J_1,J_2,J_3\}$可得 $C_{\max}=68$，排序$\{J_4,J_6,J_5,J_1,J_3,J_2\}$可得 $C_{\max}=86$，因此排序确定为$\{J_4,J_6,J_5,J_1,J_2,J_3\}$(图 1-4)。

(2)MICOT 算法

首先任选两个任务排序，计算每个机器的最大完工时间 $C(12,1)=11$，$C(12,2)=42$，$C(12,3)=49$ 依次求得所有可能排序的 $C(ab,m)$。由于 $\min C(ab,3)=C(46,3)=C(64,3)$，且 $C(46,2)<C(64,2)$，所以(4,6)为初始顺序 σ。继续计算 $C(\sigma c,m)$，按步骤继续排序，如表 1-6 所示。

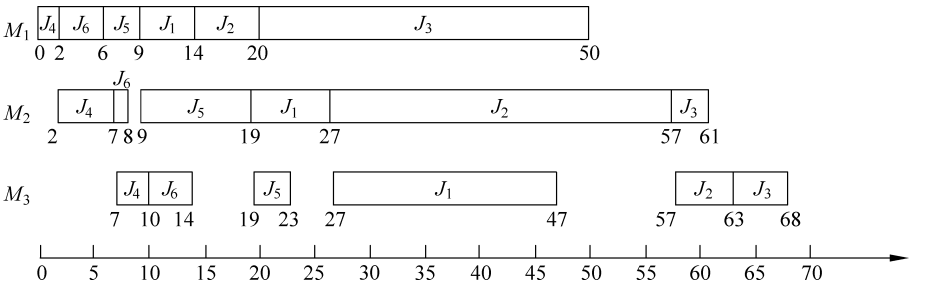

图 1-4　MINIT 算法甘特图

表 1-6　例 1-5 求解流程(3)

σ	a	$C(\sigma a,m)$			确定的顺序
		1	2	3	
46	1	11	19	39	$\sigma=465$
	2	12	42	48	
	3	36	40	45	
	5	9	19	23	
465	1	14	27	47	$\sigma=4651$
	2	15	49	55	
	3	39	43	48	

排序$\{J_4,J_6,J_5,J_1,J_2,J_3\}$可得 $C_{\max}=68$,排序$\{J_4,J_6,J_5,J_1,J_3,J_2\}$可得 $C_{\max}=86$,因此排序为$\{J_4,J_6,J_5,J_1,J_2,J_3\}$(图 1-5)。

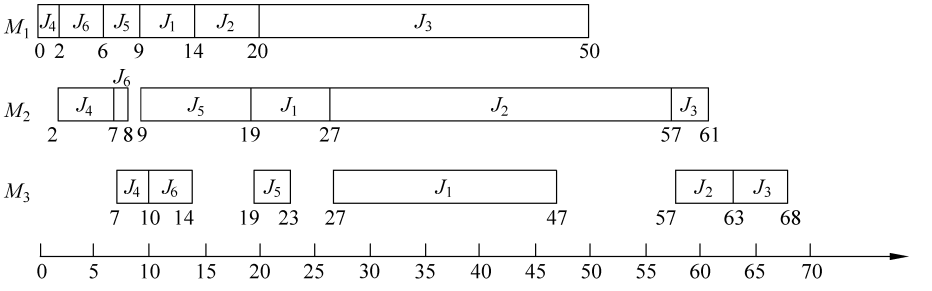

图 1-5　MICOT 算法甘特图

(3)MINIMAX 算法

首先找到最小处理时间为 1,对应任务 J_6,$k:=k+1=1$。此时,$k+k'=1<n$。最小处理时间为 30,对应任务 J_2 和任务 J_3。由于任务 J_3 在机器 3 上的处理时间比任务 J_3 小,则任务 J_3 为最后的任务,$k'=k'+1=1$。此时,$k+k'=2<n$,则继续循环。最终得到序列$\{J_6,J_4,J_5,J_1,J_2,J_3\}$(图 1-6)。

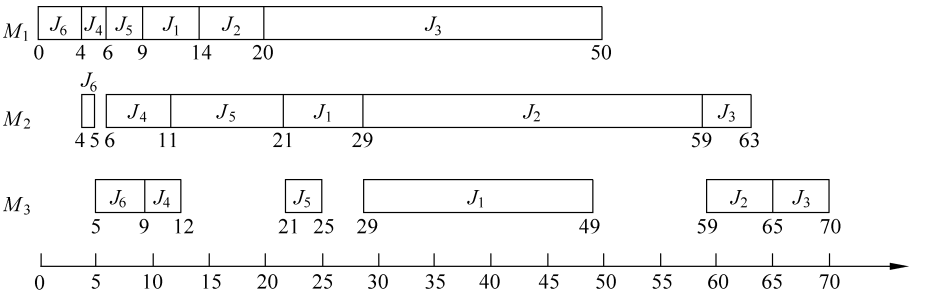

图 1-6 MINIMAX 算法甘特图

NEH 算法是一种构造性极强的启发式算法，被认为是针对流水作业现有的性能最好的启发式算法，对于置换流水作业问题有着较高的求解效率，该算法的求解时间取决于流水车间所考虑的任务数量。考虑到 n 个任务 m 台机器的情况，每个任务经过机器的顺序相同，目标是使其完工时间最小。

NEH 算法的枚举数为

$$\frac{n(n+1)}{2}-1$$

其中，有 n 个枚举是完全序列，其余的是部分序列。其步骤如下所示。

步骤 1 计算各个任务在所有机器上的总加工时间之和 $T_i=\sum_{j=1}^{m}t_{i,j}$。

步骤 2 按总加工时间之和 T_i 降序排列所有任务。

步骤 3 取出步骤 2 中产生的序列中的前两个任务，并通过计算这两个任务的完工时间来找出这两个任务的能使完工时间最小的加工顺序，在剩余步骤中，不改变这两个工作相对于对方的位置。此时设 $i=3$。

步骤 4 对于剩余任务 $i=3,4,\cdots,n$，每次将步骤 2 产生的序列中的第 i 个任务插入到步骤 3 产生的局部最优序列中的所有可能的 i 位置上，不改变已经分配的任务彼此的相对位置来实现最佳序列。此步骤的枚举数为 i。

步骤 5 如果 $i=n$，则结束；否则 $i:=i+1$，重复步骤 4。

为便于阐述 NEH 算法具体过程，现以例 1-6 进行说明。

例 1-6 考虑流水问题 $F_5\parallel C_{\max}$，其中 $n=4$，$m=5$。

	J_1	J_2	J_3	J_4
M_1	5	9	9	4
M_2	9	3	4	8
M_3	8	10	5	8
M_4	10	1	8	7
M_5	1	8	6	2

步骤 1　计算得各个任务在所有机器上的总加工时间之和为

$$T_1 = 5 + 9 + 8 + 10 + 1 = 33$$

$$T_2 = 9 + 3 + 10 + 1 + 8 = 31$$

$$T_3 = 9 + 4 + 5 + 8 + 6 = 32$$

$$T_4 = 4 + 8 + 8 + 7 + 2 = 29$$

步骤 2　得到排序$\{J_1, J_3, J_2, J_4\}$。

步骤 3　选择任务 J_1 与任务 J_3 并找到这两个任务最优的排序，如图 1-7、图 1-8 所示，显然可得$\{J_3, J_1\}$为部分最优序列。在之后的步骤中，任务 J_1 与任务 J_3 的相对位置不再改变，设此时 $i=3$。

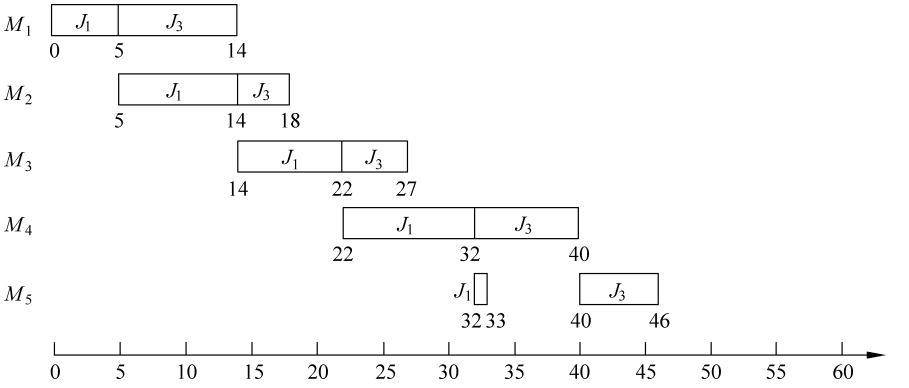

图 1-7　NEH 算法甘特图(1)

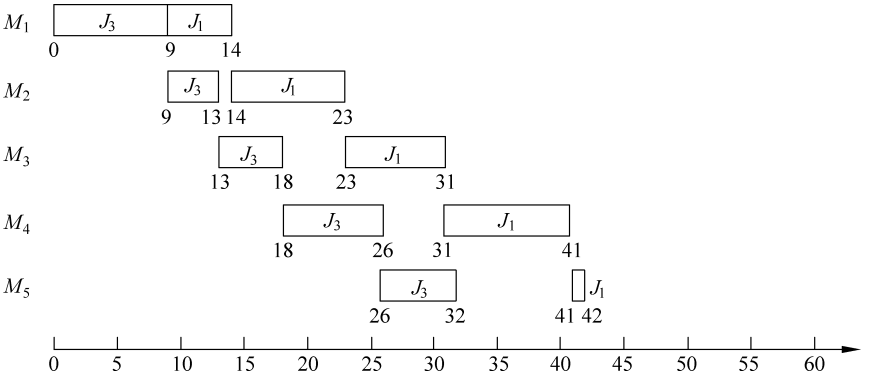

图 1-8　NEH 算法甘特图(2)

步骤 4　取步骤 2 所得序列中第 3 个位置的任务 J_2，将任务 J_2 分别放置在步骤 3 中所得部分序列的三个可能位置。如图 1-9～图 1-11 所示，可得$\{J_3, J_1, J_2\}$为部分最优序列。

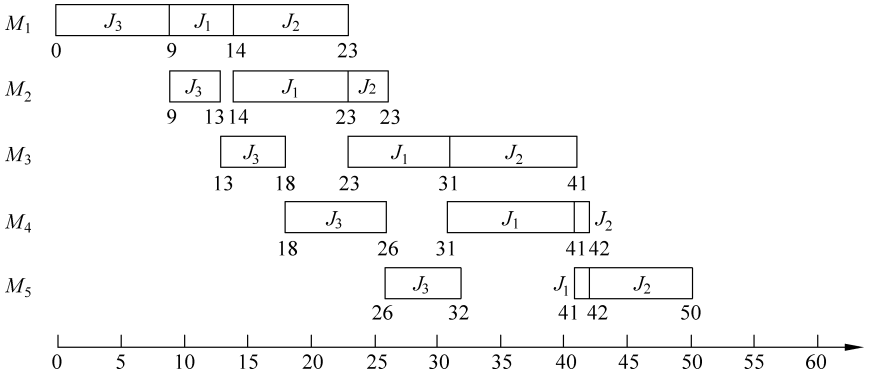

图 1-9 NEH 算法甘特图(3)

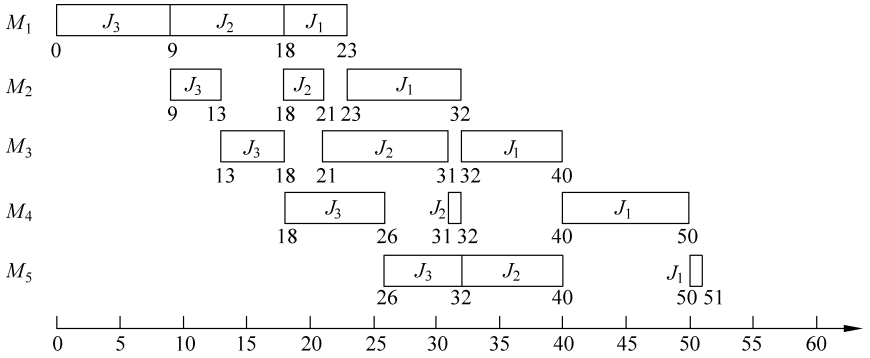

图 1-10 NEH 算法甘特图(4)

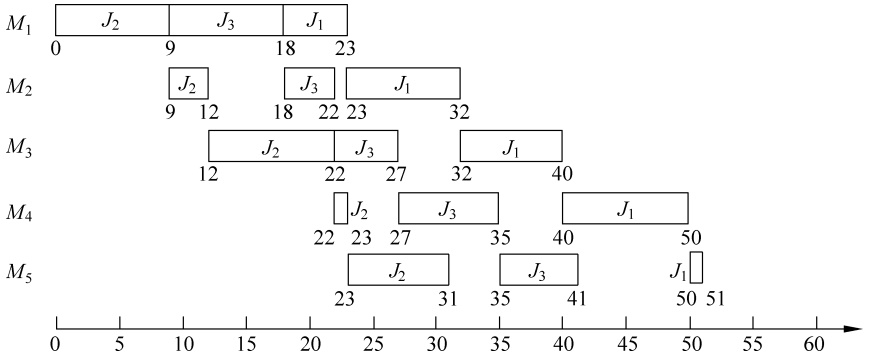

图 1-11 NEH 算法甘特图(5)

步骤 5 由于此时 $i=3$ 不为 n,因此 $i:=i+1$,跳转回步骤 4。

步骤 6 取步骤 2 所得序列中第 4 个位置的任务 J_4,将任务 J_4 分别放置

在步骤 5 中所得部分序列的四个可能位置。如图 1-12～图 1-15 所示，可得 $\{J_4, J_3, J_1, J_2\}$ 为部分最优序列。

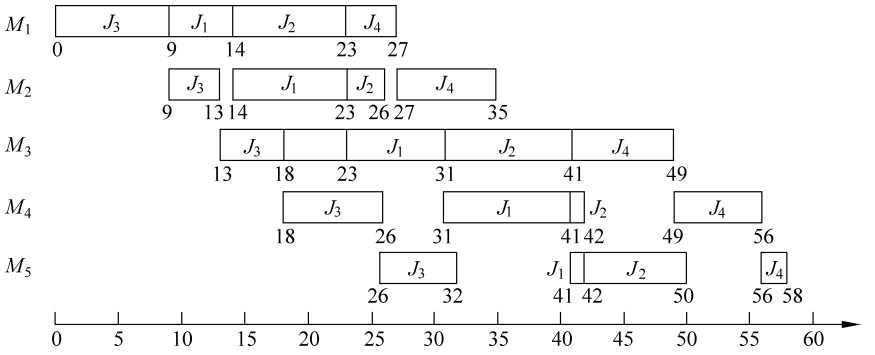

图 1-12　NEH 算法甘特图(6)

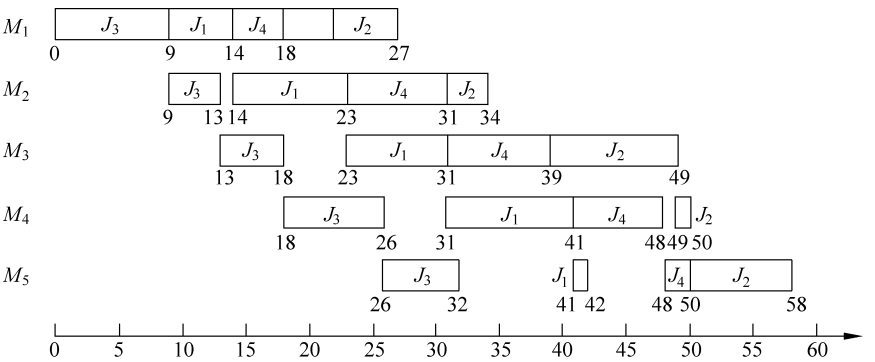

图 1-13　NEH 算法甘特图(7)

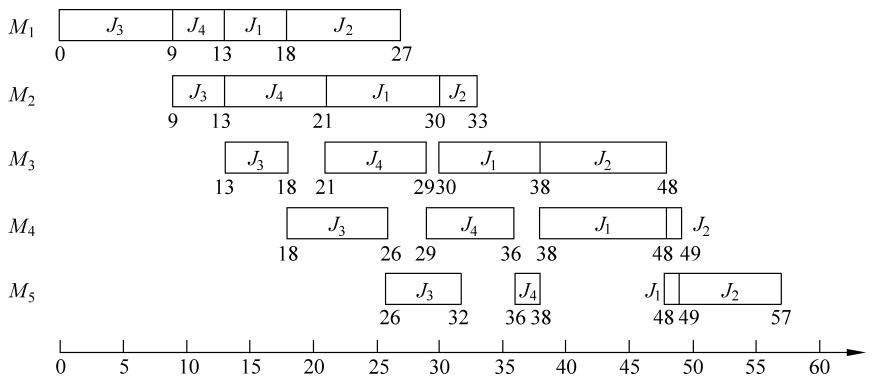

图 1-14　NEH 算法甘特图(8)

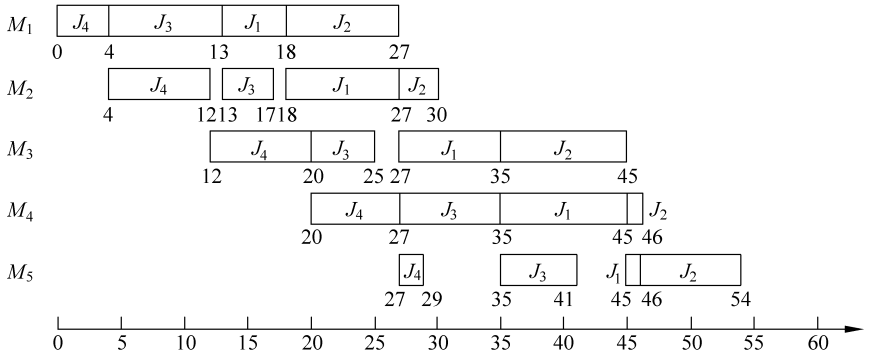

图 1-15 NEH 算法甘特图(9)

步骤 7 此时 $i=n$,停止。

通过枚举 $n!=24$ 种排序得,例 1-6 的最优序列为$\{J_4,J_3,J_1,J_2\}$。而根据 NEH 算法只需要枚举 9 次,其中 4 个是完全序列,5 个是部分序列,大大节省了计算时间。对于大型流水车间问题,可以根据 NEH 算法由计算机进行计算。

同时,对于有些 NP-难问题,还可以给出一系列近似算法$\{A_\varepsilon\}$。对于任意给定的$\varepsilon>0$,$\{A_\varepsilon\}$是多项式时间算法,而且$Z(A_\varepsilon)\leqslant(1+\varepsilon)Z(\mathrm{OPT})$,这里$Z(A_\varepsilon)$和$Z(\mathrm{OPT})$分别表示算法目标值和问题最优值,则称$\{A_\varepsilon\}$为多项式时间近似策略(polynomial time approximation scheme,PTAS)。进一步地,如果$\{A_\varepsilon\}$的时间复杂性是关于输入长度及$1/\varepsilon$的某个二元多项式,则称其为全多项式时间近似策略(fully polynomial time approximation scheme,FPTAS)。目前普遍认为,强 NP-难的问题不存在 FPTAS 算法。因此对于强 NP-难的问题,若可以设计出 PTAS 算法,则认为该问题已经解决;若不知道复杂性的某问题已经设计出了 FPTAS 算法,则此问题至少不是强 NP-难的。而对一般意义 NP-难的问题,若已经设计出了 FPTAS 算法,则也认为此类问题已经解决。

1.3 调度算法及性能分析方法

1.3.1 调度算法

求解调度问题的**算法**(algorithm),简称调度算法,指一个预先制定的执行程序,对某个调度问题的任何一个具体的例子(简称为实例),依照这个程序运行后都可以得到一个可行的排序。按照对问题输入的了解程度,调度算法可分为两类。

离线算法，问题的所有输入预先给定，要求一次性对问题做出决策，用来求解离线问题。

在线算法，输入不是预先给定，而是随时间推移逐步得到的，算法未来可能的输入不可知，在任意时刻，当有输入时，算法必须对输入及时响应，并且算法一旦做出了决策，则在后续过程中不得改变，可以用来求解在线问题，以及带有释放时间的离线问题。

1.3.2　评价算法性能的主要方法

通过 1.2 节的讨论，可知对于那些不存在多项式时间最优算法的 NP-难的问题，往往放弃寻求最优解，转而寻找一个可以快速求得的可行解，把它作为最优解的近似值。因此，自然希望得到的近似解与最优解能够最大可能地接近，以增强算法的有效性。为了从理论角度衡量确定性算法得到的近似解与最优解的接近程度，一般可以采用以下两种分析方法。

(1)**最坏性能分析**(若为在线算法，则称为竞争分析)。研究近似算法的目标函数值与离线环境下问题最优值之间的最大误差，通常采用比值的形式来衡量。对于极小化问题 π，记 I 是问题的实例，令 $Z^{A}(I)$ 表示在线算法 A 对于实例 I 的目标函数值，$Z^{*}(I)$ 表示实例 I 在离线环境下的最优值，则

$$R_{A}(\pi)=\inf\{\gamma \geqslant 1 \mid \text{对于所有的实例 } I, \text{有 } Z^{A}(I)/Z^{*}(I) \leqslant \gamma\}$$

其中，γ 称为算法 A 的最坏性能(竞争)比的上界。如果可以找到一个实例使上式中的比值取等号，或可以找到一个实例序列，使实例序列的比值收敛于 γ，则称其为紧(tight)界或最坏性能(竞争)比。最坏性能分析(竞争分析)主要是根据问题自身的特性，通过不等式的放缩求得算法目标函数值(或上界)与问题最优值(或下界)的比值。通常，每个算法的分析过程都大相径庭。对于同一个问题的不同算法，最坏性能比(竞争比)不一定相同，而且同一算法对于不同问题，最坏性能比(竞争比)也不一定相等。

一般认为最坏性能分析是一种很悲观的分析方法。用该方法评价某一算法的性能时，所研究的都是一些极端特殊的情况(或人为构造的实例)，在实际的调度过程中几乎不会发生。特别是在工业生产中，机器每天要加工数以万计的任务，根据统计的规律，任务的加工时间都会服从某种概率分布，在这种情况下，最坏情况发生的概率为零。通过数值实验分析，同样可以验证，一个性能很不理想的启发式算法，在大规模实例测试中却能够表现出收敛性。为了描述算法在求解大规模问题时的性能，可以采用下面的分析方法。

(2)**渐近性能分析**(若为在线算法，则称为渐近竞争分析)。在概率极限意义下研究近似算法的目标函数值与离线环境下问题最优值之间的接近程度，通

常采用比值的形式来衡量。对于极小化问题 π，记 I 是问题的实例，令 $Z^{A}(I)$ 表示算法 A 对于实例 I 的目标函数值，$Z^{*}(I)$ 表示实例 I 在离线环境下的最优值，n 表示问题规模，n_0 表示一个给定的数值，则

$$R_{A}^{\infty}(\pi)=\inf\{\gamma\geqslant 1 \mid \text{存在 } n_0,\text{对于所有 } n\geqslant n_0 \text{ 的实例 } I,\ \text{有 } Z^{A}(I)/Z^{*}(I)\leqslant\gamma\}$$

称为算法 A 的渐近性能(竞争)比。如果算法的渐近性能(竞争)比为 1，则称其具有渐近最优性(收敛性)。若某算法是渐近最优的，那么极限意义下(理想状态)，该算法得到的可行解与问题的最优解是等价的。

在大规模工业生产环境中，算法如果具备了收敛性，就可以直接作为最优算法使用，能够为企业节省大量的计算时间。同时，有了收敛性的保证，可以进一步对算法进行改进，以期在合理的时间内获得最好的解。虽然渐近性能分析在理论与应用方面比其他方法更具优势，但是在具体分析过程中需要用到如概率分析和随机过程等复杂的数学手段。因此，即使对一些很简单的算法进行渐近性能分析，也会面临相当大的挑战。

1.4 相关调度问题研究现状

Johnson(1954)发表了第一篇关于流水车间的学术论文，证明了 $F_2\parallel C_{\max}$ 问题可以通过著名的约翰逊规则在多项时间内求得最优解，这也是第一篇关于调度理论的研究论文。在流水车间中，除 $F_2\parallel C_{\max}$、$F_2\mid \text{prmp}\mid C_{\max}$(Gonzalez et al,1978)、$F_2\mid \text{nwt}\mid C_{\max}$(Gilmore et al,1964)及少数多项式可解的特殊情况之外，其余都是 NP-难问题，而且绝大多数都具有强 NP-难性质。因此，精确算法仅围绕小规模问题讨论。Tomazella 等(2020)在流水作业调度问题中对分支定界算法进行了全面回顾并提出了进一步研究的指南和方向。而针对中等规模和较大规模的问题，主要的研究工作围绕构造启发式算法，运用智能优化算法及评价启发式算法的性能展开。Rubén 等(2005)通过基准集对 25 种启发式和元启发式算法进行了比较和评估，在不同算法的效率和有效性方面得到了具有参考性的结论。

近年来，流水作业调度相关领域的研究受到学界的广泛关注。Rossit 等(2018)综述了非置换流水作业调度问题的研究现状。Miyata 等(2019)详细阐述了带有阻塞约束的流水作业调度问题，全面总结了相关研究进展。Komaki 等(2019)综述了装配流水作业模型的求解方法及其在工业中的应用。Sun 等(2020)回顾了考虑位置相关学习效应的流水作业调度问题，目标为极小化总完工时间加权和。随着代工生产的逐渐发展，多代理(multi-agent)调度成为热门

的话题。以前的研究大多集中于单机情况，Bai 等(2022)将多代理问题推广至流水作业调度中，取得了突破性进展。为了更加贴近现代制造行业中的实际生产环境，混合流水调度逐渐引起学者们的注意，王凌等(2011)对混合流水线调度的研究进展做出了整合。此外，随着全球经济和分散经济的发展，大型制造企业出现了分布式生产，分布式流水车间的研究也引起了新的关注(Li et al，2021)。

在优化目标方面，多数研究针对完工时间相关的经济目标进行优化，如最大完工时间、完工时间和、提前与拖期、误工损失等。此类经济目标优化问题的相关研究大多是单目标优化问题，少数研究考虑了多目标优化问题。然而，在世界范围内不断增加的碳排放量正在加剧全球变暖问题，许多国家和国际组织已经开始关注这一问题，并建立了减少碳排放的机制，而且制造企业也越来越关注节能生产，因此，制造业的绿色调度已成为一个研究热点(Chen et al，2020)。相关研究大多同时考虑了经济目标和绿色目标，如总能耗等，此类问题是一类复杂程度较高的多目标优化问题。

1.4.1　渐近分析研究现状

渐近性能分析最初是用来研究求解装箱(bin packing)问题的近似算法(Coffman et al，1991)，之后被引入调度算法理论分析领域。目前以所能查阅的文献来看，只有极少数学者从事近似算法的渐近性能研究，该方法的研究成果主要还是集中于确定性调度问题。本书将流水作业模型相关研究内容阐述如下。

目前调度算法及其渐近分析的相关研究主要关注经典的流水作业模型。Bai 等(2010)研究了 $F_m|r_j,(\text{online})|C_{\max}$ 问题，证明了先到先服务(first-come，first-served，FCFS)规则的渐近最优性。针对 $F_m\|\sum w_jC_j$ 问题，Kaminsky 等(1998)证明其最优值渐近地等于相应的 $1\|\sum w_jC_j$ 问题最优值，并指出 WSPT 算法的渐近最优性，而且他们还讨论了求解 $F_m\|\sum C_j$ 问题的 SPT 算法的渐近最优性(Kaminsky et al，2001)。Xia 等(2000)在鞅论的基础上给出了 SPT 算法渐近最优性的另一种证明方法，简化了 Kaminsky 等(2001)的证明。针对 $F_m|r_j|\sum w_jC_j$ 问题，根据单机 WSPTA 算法的思想，Liu 等(2005)分别给出了离线和在线近似算法，并证明了此类算法的渐近最优性。针对 $F_m\|\sum C_j^2$ 问题，Koulamas 等(2005)证明了 SPT 算法的渐近最优性。白丹宇(2015)研究了 $F_m|r_j,(\text{online})|\sum C_j^2$ 问题，在工件释放时间与问题规模同阶的假设下，

证明了两个基于 SPTA 规则近似算法的渐近最优性。之后，Bai 等(2014)将上述结论推广至 $F_m|r_j|\sum C_j^k$ ($k>2$)问题。针对 $F_m \parallel D_{\max}$ 和 $F_m|r_j|D_{\max}$ ($D_{\max}$ 为最大运输完工时间)问题，Kaminsky(2003)分别提出了基于最长运送时间(LDT)规则的近似算法，并证明了它们的渐近最优性。Iravani 等(2005)研究了单服务器流水车间问题(所有阶段的操作都由一台机器来完成)，目标函数是极小化调整和持有费用之和，在持有费用与阶段编号一致非减的假设下，证明了一类具有链式结构算法的渐近最优性。Chen 等(2007)研究了 $F_m|\text{online}, p_{i,j}\sim\text{sto}|\sum w_jC_j$ (sto 表示随机)问题，在工件权重有界且加工时间为独立同分布随机变量的假设下，证明了任意的非延迟(non-delay)算法都具有渐近最优性。针对 $F_m|\text{prmu},r_j|\sum C_j$ 问题，Li 等(2018)设计了 GSH 构造启发式，其利用 SSH 启发式产生初始解，设计影响力计算公式选出对目标值影响力较高的作业进行重新插入操作，并验证了该算法在作业是否同时到达的两种情况下均保持了求解质量和计算时间的平衡性。针对 $F_m|\text{prmu},\text{ST}_{\text{sd}}|C_{\max}$ (ST_{sd} 表示顺序相关准备时间，sequence-dependent setup times)问题，Sioud 等(2018)提出了一种基于准备时间的启发式算法。Bai 等(2018)研究了 $F_m|r_j,\text{le}|C_{\max}/\sum C_j/\sum C_j^2$ 问题，证明了一类基于 SPT 规则算法的渐近最优性。之后，针对 $F_m|\text{le}, r_j|L_{\max}$ 问题，Bai 等(2021)证明了基于最早交付日期优先(earliest due date available，EDDA)规则的渐近最优性。针对 $F_m|r_j|C_{\max}^a+\theta C_{\max}^b$ (目标函数为极小化两个代理的加权最大完工时间之和)问题，Bai 等(2022)设计了基于优势代理优先的启发式，并证明其渐近最优性。针对 $F_m|\text{prmu}|F_l(C_{\max},\sum C_j)$ (F_l 表示两个目标的线性函数)问题，Li 等(2019)提出了基于当前和未来偏差的构造启发式，分别使用两组权重解决完成时间上下界耦合偏差问题和双目标权衡问题。Wang 等(2019)提出了分别基于 ARB、SPT、NEH 和 FL(Framinan and Leisten)规则的启发式，并验证了 NEH 和 FL 对于极小化最大完工时间问题的求解效果最好。而针对极小化加权完工时间和问题，FL 的求解效率最高。针对 $F_m|\text{nwt}|C_{\max}+\theta\sum C_j$ (目标函数为极小化最大完工时间与完工时间和的加权之和)问题，Ye 等(2020)提出了一种权衡平衡启发式，其中引入了因式分解方法来构造基于当前和未来空闲时间的初始排序，并提出了一种新的估计方法进行不同目标的归一化。Puka(2021)针对 $F_m|\text{prmu}|C_{\max}$ 问题提出 N-NEH+算法，该算法提供了一个部分排序的候选列表，其中每个候选排序均包含 N 个任务，具有相等或相似的目标值。针对 $F_m|\text{prmu},\text{le}|\sum w_jC_j$ 问题，其中学习效应与工件加工位置相关，Sun 等(2021)分别提出了基于 ARB

(arbitrary busy)规则和加权最短处理时间优先规则(weighted shortest processing time first)的两种启发式,并分析了它们的最坏情况误差界。针对工件具有对数截断学习效应的 $F_m|\text{prmu},\text{le}|C_{\max}/\sum w_jC_j$ 问题,针对具有场景相关加工时间的两阶段 $\text{AF}_3\|C_{\max}$(AF 表示装配流水车间"assembly flow shop",装配流水车间包括两个阶段:加工阶段和装配阶段。加工和装配阶段均由一台或多台并行工作的机器组成。在加工阶段生产加工产品的不同组件,然后在装配阶段的装配机上组装形成最终产品)问题,Wu 等(2021a)提出了一系列基于约翰逊规则的启发式寻找鲁棒近优解,其中有些使用了成对交换方法。Wu 等(2021c)针对同时优化最大完工时间和机器利用率的阻塞流水车间调度问题,提出了结合峰型拟合方法的 NEH 启发式,其中设计了基于作业处理时间四分位偏差的优先级规则和断链规则。

少数研究关注了其他流水作业模型,包括混合流水车间、分布式流水车间和分布式混合流水车间。Koulamas 等(2000)研究了两阶段和三阶段 HF $(P_{m1},P_{m2})/(P_{m1},P_{m2},P_{m3})\|C_{\max}$(HF 表示混合流水车间 hybrid flow shop)问题,设计了近似算法并证明了其渐近最优性;然后又将结论推广至 k 阶段的一般情况。之后,Kyparisis 等(2006)又将前面的近似算法进行了推广,用于解决多阶段 HF$(Q_{m1},Q_{m2},\cdots,Q_{mk})\|C_{\max}$ 问题,并对其进行了渐近性能分析,得出了渐近最优的结论。针对 $\text{DF}_m|\text{prmu}|\sum C_j$(DF 表示分布式流水车间 distributed flow shop。分布式生产系统充分利用了不同国家和地区间多个工厂的资源,其通过原材料的有效分配、生产力的最优组合以及科学合理的资源共享等方式,以较低的成本实现产品的快速生产与制造)问题,Fernandez 等(2018)通过分析问题性质得到三个支配规则,设计了一类基于 NEH 的启发式,其中提出了两种解的表示方法和六种工厂分配规则,并设计了两种加速程序用于减少目标值的计算次数。针对 $\text{DHF}|\text{block},\text{prmu}|C_{\max}$(DHF 表示分布式混合流水车间 distributed hybrid flow shop),Shao(2021)提出了改进 NEH 启发式,并设计插入改进策略对部分排序重新进行优化。

1.4.2　流水作业调度问题研究现状

求解流水作业调度问题目前大多采用"精确算法"和"智能优化"两类优化方法,下面分别综述各自的研究现状与发展动态。

1. 精确算法

1993 年,Karabati 等(1993)提出了基于拉格朗日松弛下界的分支定界算法求解生产调度问题,相关结论可以推广至其他以完工时间和作为目标的调度模

型中。根据文献来看，应用精确算法求解流水作业调度问题的现有研究主要关注经典的流水作业模型，少量研究关注混合流水作业模型、装配流水作业模型和分布式流水作业模型，相关算法大多应用分支定界、动态规划、数学规划算法，相关内容阐述如下。

1）流水作业模型

（1）经济目标

① 双机

针对 $F_2 \| \sum Y_j$（目标函数为总误工损失“total late work”，其中任务的误工损失是指对其交付期之后处理部分进行的惩罚）问题，Chen 等（2019）基于问题松弛版本 $F_2 | d_j = d | \sum Y_j$（$d_j = d$ 表示所有任务具有公共交付期 d）的枚举求解方法，设计了分支定界算法，其中分别利用遗传算法和列表启发式确定初始上界和改进上界值，并提出了优势规则和新下界提高算法计算效率。之后，Chen 等（2022）总结了 $F_2 | d_j = d | \max\left\{\sum X_j\right\}$（目标函数为极大化总提前任务数“total early work”，其中任务的提前任务数 X_j 指其交付期之前已处理完成的总任务数）问题的两个最优解性质用于设计动态规划算法，该算法只关注第二台机器上的任务分配，时间复杂度为 $O(n^2 d^2)$。针对具有非强制性空闲时间的 $F_2 | \text{prmu} | \sum (E_j + T_j)$（目标函数为极小化提前和拖期总和）问题，Schaller 等（2019）通过插入空闲时间减少部分工件的提前，证明了仅第二台机器需要插入空闲时间。他们推广了单机问题的下界和支配条件，并提出了两个分支定界算法进行求解。之后，针对 $F_2 | \text{nwt} | \sum (E_j + T_j)$ 问题，Schaller 等（2020）提出分支定界算法，设计了基于 EDD 的下界。针对考虑紧急任务的 $F_2 \left| r_{i \in J_A} \geqslant 0 \right| \alpha \cdot \left(\sum_{i \in J_A} T_i \right) + (1 - \alpha) \cdot (\max_{i \in J_B} C_i) \sqrt{a^2 + b^2}$（任务根据紧急程度可被分为两类，紧急任务 A 类和正常（非紧急）任务 B 类。实际生产制造过程希望紧急任务尽可能在无须等待的情况下被处理。J_A 和 J_B 分别表示 A、B 两类任务的集合，目标函数为极小化非紧急任务最大完成时间与紧急任务总拖期的加权和）问题，Jeong 等（2020）证明了该问题的一些支配性质，提出了一种分支定界算法，其中利用两种列表调度算法和 4 种基于 NEH 或 FL 的构造启发式获得初始上界，并基于支配性质设计了剪支规则和下界。针对 $F_2 | l_j | C_{\max}$（l_j 表示时间延迟）问题，Mkadem 等（2021）提出了分支定界方法，扩展了搜索树中使用的最佳下界，并提出了基于局部搜索的上界和三个支配规则。针对 $F_2 | \text{nwt}, \text{prpt} | \text{TADC}$（prpt 表示比例，在比例流水车间中，每个作业的处理时间在所有机器上均相

等;目标函数为作业完成时间的绝对偏差总和“total absolute deviation of job completion times”),Kovalev 等(2019)证明存在一个关于作业处理时间的半 V 形最优序列,并提出了一个 $O(n^3)$时间的动态规划算法进行求解。

② 多机

Shang 等(2018)针对 $F_3 \parallel C_{\max}$ 设计了动态规划算法,提出了一些支配条件。Kim 等(2019)针对具有重叠等待时间约束的 $F_3 \mid w_{i1}, w_{i2} \mid C_{\max}$(在第一台机器上完成加工的工件,必须在特定时间段内在第二台和第三台机器上进行处理。第一台和第二台机器之间的等待时间与第一台和第三台机器之间的时间存在重叠的现象,称为重叠的等待时间约束。任务 i 在前两台机器之间、第一台和第三台机器之间的等待时间分别受限于两个时间限制 w_{i1} 和 w_{i2})问题提出了分支定界算法进行求解,其通过分析重叠等待时间约束来证明问题的一些支配性质,以减少问题的搜索空间,并推导了 7 个下界。

针对 $F_m \mid r_j, \text{le} \mid C_{\max} / \sum C_j / \sum C_j^2$ 问题,Bai 等(2018)设计了分支定界算法,其中提出有效的分支规则和基于最短剩余处理时间优先规则的下界,加速算法收敛。之后,Bai 等(2021)针对 $F_m \mid \text{le}, r_j \mid L_{\max}$ 问题设计了分支定界算法,其中提出了有效的分支规则和基于可中断 EDD 规则的下界。针对 $F_m \mid r_j \mid C_{\max}^a + \theta C_{\max}^b$ 问题,Bai 等(2022)设计了分支定界算法,其中提出基于释放日期的分支规则和基于可中断的下界。针对工件具有对数截断学习效应的 $F_m \mid \text{prmu}, \text{le} \mid C_{\max} / \sum w_j C_j$ 问题,Wang 等(2019)提出了分支定界算法,并推导了问题的两个下界。Gerstl 等(2019)研究了 $F_m \parallel \sum Y_j$,证明了一个最优解性质,针对该问题的两个版本分别设计有效的伪多项式动态规划算法,并验证其在大规模算例上的性能。Mor 等(2019)针对 $F_m \mid \text{prpt}, \text{rej} \mid C_{\max} : \sum_{j \in \overline{A}} e_j \leqslant \text{E}$ 和 $\text{F}_m \mid \text{prpt}, \text{rej} \mid \sum_{j \in \overline{A}} e_j : \sum_{j \in A} p_j \leqslant K$(rej 表示工件可拒绝,$A$ 表示被接收工件的集合,$\overline{A}$ 表示被拒绝工件的集合)两个问题提出改进伪多项式动态规划算法进行求解。针对无等待置换流水车间调度,其中优化目标为最小化剩余工作内容(由于缺乏可用资源而无法完成的工作量)的条件风险价值(一种风险损失度量,用于量化不确定性因素所产生的风险),Urgo(2019)提出了分支定界算法获得问题的鲁棒解。针对具有简单线性退化效应的 $F_m \mid p_{ij} = b_{ij} t \mid \log C_{\max} / \sum \log C_j / \sum w_j \log C_j / \sum \log^2 C_j$(目标函数分别是最大完工时间的对数、完工时间的总对数、完工时间的总加权对数和完工时间的二次对数之和。这里的 p_{ij} 为实际处理时间,$b_{ij} > 0$ 为退化率)问题,Sun 等(2019)提出了分支定界算法,分别为 4 个优化目标

设计了下界。针对部分处理机零空闲的极小化总工期流水作业问题，Bektas 等(2020)采用 Benders 分解算法进行精确求解，提出基于邻域搜索的割生成算法，加速算法收敛。针对极小化机器空闲时间的置换流水车间调度问题，Liou(2020)探讨了正负机器空闲时间对流水车间的最大完工时间的影响，并提出了分支定界算法进行求解，其中设计了基于机器空闲时间的下界，提出一个与前后顺序无关的支配条件，以提高算法的运算效率。针对 $F_m|\text{prmu}|C_{\max}$ 问题，Gmys 等(2020)提出分支定界算法进行求解，该算法可在多核 CPU 上顺序或并行地进行树搜索，并通过结合动态分支和基于在线学习启发机制的下界，设计了新的节点分解方法。Mor 等(2020)针对 $F_m|\text{prpt},\text{rej},\text{le}|C_{\max}/\sum T_j/\sum_m^{i=1}C_{\max}^{(i)}:\sum_{j\in R}e_j\leqslant E$(目标函数表示在总拒绝成本 $\sum_{j\in R}e_j\leqslant E$ 的条件下分别极小化 $C_{\max}$、$\sum T_j$ 和 $\sum_m^{i=1}C_{\max}^{(i)}$。其中 R 表示被拒绝工件的集合，$\sum_m^{i=1}C_{\max}^{(i)}$ 表示总负载，即所有机器上最后一个作业的完工时间总和)问题提出了伪多项式动态规划算法进行求解。Nagano 等(2020)针对 $F_m|\text{block},\text{prmu}|\sum C_j$ 问题提出改进分支定界算法，其中设计了基于机器空闲和阻塞的下界。Koulamas(2020)针对比例 $F_m|\text{prpt}|\sum T_j$ 问题提出了一种伪多项式动态规划算法。之后，Koulamas 等(2021)针对工件可拒绝的 $F_m|\text{nwt}|C_{\max}+\sum e_j$ ($\sum e_j$ 为总拒绝成本)问题提出了动态规划算法。Ren 等(2021)针对 $F_m|r_j|C_{\max}/D_{\max}$ 问题设计了分支定界算法进行精确求解，其中设计了下界和分支规则，显著地缩小了搜索空间。针对 $F_m|\text{prmu},\text{le}|\sum w_jC_j$ 问题，其中学习效应与工件加工位置相关，Sun(2021)等分别提出了分支定界算法，设计了 4 个下界。针对带有不确定处理时间(处理时间在给定的时间间隔内变化)的鲁棒流水车间问题，Levorato 等(2022)扩展了确定性情况下的两个经典混合整数规划公式，并将其与列约束生成框架相结合，获得该问题在最坏场景下极小化最大完工时间的鲁棒解。此外，还开发了一种多项式时间动态规划算法，用于确定该问题的最坏场景。

(2) 绿色目标

针对具有外包选择(作业外包用于解决生产能力不能满足需求的问题)的准时制节能 $F_m,R_m\|(\text{TC},\text{TEC})$(TC 表示生产总成本“total cost”，TEC 表示总能耗“total energy consumption”)问题，Tirkolaee 等(2020)提出了基于 ε 约束的多目标模糊数学规划技术，将所提出的双目标模型视为单目标混合整数线性规划模型，然后引入自适应人工鱼群算法辅助提供帕累托最优解。针对考虑

能源循环利用的双目标随机优化绿色流水作业调度问题，Wang 等(2020b)设计了多目标数学规划算法，引入 L-shaped 方法更加高效地切割值域空间，降低算法迭代次数。

2) 其他流水作业模型

针对考虑能耗与经济指标的自动化码头调度问题，Xin 等(2014)利用两阶段算法将场景分解为宏观层面和微观层面。其中宏观层面将该问题抽象为极小化总工期的柔性流水作业调度问题，利用商业优化软件 CPLEX 求解；微观层面以极小化总工期为前提求解最小能耗的设备运作方案。针对考虑累积学习效应的两阶段 $\mathrm{AF}_3|\mathrm{le}|\sum C_j$ 问题，Wu 等(2018a)提出分支定界算法，推导了支配性质和三个下界。针对具有位置相关学习效应的两阶段 $\mathrm{AF}_3|\mathrm{le}|C_{\max}$ 问题，Wu 等(2018b)提出了分支定界算法，推导了一些支配性质和一个下界。针对考虑位置相关学习效应的两阶段双代理 $\mathrm{AF}_3|\mathrm{le}|\sum_{j\in A}C_{3j}^{A}:\sum_{j\in B}C_{3j}^{B}<U_B$ (目标函数表示在第二个代理总完成时间给定上界的前提下，最小化第一个代理的总完成时间)问题，Wu 等(2019)推导了一些支配命题和三个下界，并用于设计分支定界算法。针对两阶段 $\mathrm{AF}_3|\mathrm{le}|\sum C_j$ 问题，Wu 等(2020)提出了分支定界算法，证明了机器在有无空闲时间两种情况下的 10 个支配性质，并推导了下界公式。针对具有场景相关加工时间的两阶段 $\mathrm{AF}_3\|C_{\max}$ 问题，Wu 等(2021a)提出了分支定界算法获得鲁棒解，其中推导了一个支配性质和一个下界。针对具有截断学习效应的两阶段 $\mathrm{AF}_3|\mathrm{le}|C_{\max}$ 问题，Wu 等(2021b)推导了支配规则、引理和一个下界，并将其应用于分支定界算法的设计。针对极小化总工期的分布式流水作业调度问题，Hamzadayi(2020)设计了 Benders 分解算法，提出基于问题特征的割生成算法，加速算法收敛。

2. 智能优化算法

智能优化方法是一类基于计算智能机制求解复杂优化问题的算法总称，能够在限定时间内获得作业调度问题的满意解。目前，应用智能优化算法求解流水作业调度问题的研究成果较多，主要关注经典流水作业、混合流水作业、装配流水作业和分布式流水作业及其组合模型，相关算法不仅包括遗传算法、蚁群算法、禁忌搜索算法等经典算法，还有近年来出现的一些新型智能优化算法，相关内容阐述如下。

1) 流水作业模型

(1) 经济目标

针对 $F_m|\mathrm{nwt}|C_{\max}$ 问题，Engin 等(2018)设计了混合蚁群算法，该算法通

过引入遗传算法的交叉和变异机制跳出局部极小值，进行 5 种交叉机制（单点交叉、基于位置的交叉、次序交叉、部分映射交叉、线性次序交叉）和 5 种变异机制（反转变异、邻居交换变异、互换变异、插入变异、三联体变异）的组合实验，验证了基于位置的交叉和反转变异是求解该问题的最佳组合。针对同时考虑退化和学习效应的随机流水车间调度问题（作业的处理时间不确定且依赖于时间），Fu 等（2018）同时优化最大完工时间和总拖期两个目标，并开发了新型多目标离散烟花优化算法。该算法结合迭代搜索方法和学习搜索方法改进爆炸操作，其中基于模拟退火的迭代搜索方法可对种群中的较优解进行进一步优化，学习搜索方法用于引导劣解的优化方向。针对 $F_m|\mathrm{prmu},\mathrm{ST_{sd}}|C_{\max}$ 问题，Sioud 等（2018）提出了改进候鸟算法，提出基于交换和向前插入移动的自适应邻域搜索，引入禁忌表、重启机制，并设计考虑迭代次数的领飞鸟选择方法用于跳出局部最优。针对 $F_m|\mathrm{block}|\sum T_j$ 问题，Ribas 等（2019）提出了迭代贪婪算法，设计了几种用于生成初始解的构造性启发式，并引入变邻域搜索调整作业的车间分配。针对 $F_m|\mathrm{nwt}|C_{\max}$ 问题，Zhao 等（2019）提出了具有变邻域搜索机制的混合生物地理学优化算法，其利用改进的 NEH 和最近邻机制生成初始种群。为了加快算法的收敛速度，他们设计了一种混合迁移算子，该算子结合了路径重链技术和基于块的自改进策略，并将迭代贪婪算法引入变异算子中，在开发阶段得到了很好的解；此外，他们还设计了一种基于块邻域结构和插入邻域结构的变邻域搜索策略，在每次迭代中围绕当前最优解进行局部搜索，并引入精英策略保留当前种群中的最优解。之后，针对 $F_m|\mathrm{prmu},\mathrm{nwt}|\sum T_j$ 问题，Zhao 等（2020b）提出了混合离散水波优化算法。为了提高初始种群的质量，他们提出了一种基于新的优先级规则和改进 NEH 启发式的初始化方法。在传播阶段，引入了一种具有 11 种不同邻域结构的自适应选择邻域搜索策略，扩大了搜索范围，实现了探索与开发的平衡。在中断阶段，提出了一种基于变邻域搜索的局部强化方法搜索当前最优解周围的区域，并保持种群的多样性。在折射阶段，为了避免算法陷入局部最优，引入了扰动策略。针对 $F_m|\mathrm{le},r_j|L_{\max}$ 问题，Bai 等（2021）设计了离散人工蜂群算法，其中提出基于 EDDA 规则的初始化方法和基于关键间隔邻域算子的混合邻域搜索机制，分别用于提高初始种群和最终解的质量。之后，针对 $F_m|r_j|C^a_{\max}+\theta C^b_{\max}$ 问题，Bai 等（2022）设计了离散人工蜂群算法，并提出 5 种邻域搜索策略改进雇佣蜂和跟随蜂阶段。针对考虑设备维护的 $F_m|\mathrm{prmu}|C_{\max}/\mathrm{ETP}$（ETP 表示提前/拖期惩罚“earliness tardiness penalties”）问题，Branda 等（2021）设计了带有精英保留策略的遗传算法，提出部分匹配交叉策略消除不满足问题约束的子代个体，并验证了该算法在不同问题规模上的性能；此外，他们还利用以上搜索过程得到的非支配解构建

并更新外部档案库，且基于外部档案库改进了选择策略。Libralesso 等(2021)研究了 $F_m|\text{prmu}|C_{\max}/\sum C_j$ 问题，并提出了迭代波束搜索算法，其分支策略引入了前向分支和双向分支。针对 $F_m|r_j|C_{\max}/D_{\max}$ 问题，Ren 等(2021)使用混合离散差分进化算法进行求解，提出基于种群划分的初始化方法，并设计基于插入邻域的破坏-重建策略，防止算法陷入局部最优。

(2) 绿色目标

针对运输时间可控的 $F_m|\text{prmu},\text{ST}_{\text{sd}}|(C_{\max},\text{TEC})$ 问题，Jiang 等(2019)改进了基于分解的多目标进化算法(multi-objective evolutionary algorithm based on decomposition，MOEA/D)进行求解，其中利用动态匹配策略进行问题分解和子问题求解。通过分析问题性质，提出了两种启发式生成总准备时间较小的解，用于设计局部搜索强化策略。针对考虑能效的 $F_m|\text{block},\text{ST}_{\text{sd}}|(C_{\max},\text{TEC})$ 问题，Han 等(2020)设计了一种离散进化多目标优化算法，其中采用可变单目标的启发式用于种群初始化，设计两种自适应开发和自适应探索的进化算子生成高质量的解，并提出了一种基于惩罚区间的插入局部搜索算法提高算法的开发能力。针对 $F_m|\text{prmu}|(\text{TF},\text{TEC})$ (TF 表示总流经时间“total flow time”)问题，Öztop 等(2020)提出了两种多目标迭代贪婪算法和一种多目标可变块插入算法，利用速度缩放方法降低能耗，并设计了相应的多染色体编码方法，包括一个作业向量和一个速度向量。针对存在动态速度缩放技术 $F_m|\text{nwt},\text{prmu}|(C_{\max},\text{TEC})$ 问题，Wu 等(2020)提出了自适应多目标变邻域搜索算法。通过分析问题非支配解的结构性质，开发了两个基本的速度调整启发式算法，可以在最大完工时间不会变长的前提下，降低解的总能耗；此外，还引入变邻域下降策略，并采用一种自适应机制来动态地确定邻域结构。为了进一步提高算法的性能，设计了扰动策略和加速策略。针对 $F_m|\text{nwt}|(C_{\max},\text{TEC})$ 问题，Zhao 等(2020a)分析了问题的性质，提出具有问题特定知识的两阶段协同进化算法，该算法使用两个构造型启发式算法生成初始解。在算法第一阶段中，使用迭代局部搜索策略搜索潜在的极值解，并设计了混合邻域结构用于改善解的质量；在第二阶段中，提出基于关键路径知识的变异策略用于扩散极值解到整个已经找到的帕累托前沿面；在算法的迭代过程中，利用迭代局部搜索和变异策略形成了一个协同进化的闭环系统，并缩小了问题的搜索空间。针对考虑能效的两阶段 $F_2|\text{prmu},r_j,p\text{-batch},\text{block}|(C_{\max},\text{TEC})$ (p-batch 表示并行批处理机)问题，Zheng 等(2020)提出了多目标混合蚁群优化算法，其中有最大-最小信息素限制规则和局部搜索规则，分别用于避免陷入局部最优和增强邻域搜索能力，并采用了两种分别基于 FIFO 和约翰逊规则的批处理调度规则。

2）装配流水作业模型

针对考虑累积学习效应的两阶段 $AF_3|le|\sum C_j$ 问题，Wu 等(2018a)提出了具有不同粒子速度和粒子位置更新公式的 6 种混合粒子群算法，并通过实验分析了速度和惯性权重之间的关系。针对具有位置相关学习效应的两阶段 $AF_3|le|C_{max}$ 问题，Wu 等(2018b)提出了 3 种基于云理论的模拟退火算法。针对考虑位置相关学习效应的两阶段双代理 $AF_3|le|\sum_{j\in A}C_{3j}^{A}:\sum_{j\in B}C_{3j}^{B}<U_B$ 问题，Wu 等(2019)提出了 4 种基于不同改进策略的混合粒子群优化算法。针对两阶段 $AF_3|le|\sum C_j$ 问题，Wu 等(2020)提出了 4 种元启发式算法，包括遗传算法、基于云理论的模拟退火算法、人工蜂群算法和迭代贪婪算法，其中引入迭代局部搜索策略的迭代贪婪算法的性能最优。针对具有场景相关加工时间的两阶段 $AF_3\|C_{max}$ 问题，Wu 等(2021a)提出了 4 个基于云理论的模拟退火超启发式算法获得鲁棒近优解，其中结合了 7 种邻域算子。针对具有截断学习效应的两阶段 $AF_3|le|C_{max}$ 问题，Wu 等(2021b)提出了动态差分进化算法、混合贪婪迭代算法和遗传算法，验证了动态差分进化算法在小规模实例上的性能更优，而混合贪婪迭代算法在大规模实例上的性能更优。针对考虑预防性维护和纠正性维护的 $AF_m|prmu|(C_{max},TMC)$(TMC 为预防性维护和纠正性维护的总维修成本“total maintenance costs”)，Zhang 等(2021)提出了考虑维护的评价指标，设计了基于预防性维护策略的重启迭代 Pareto 贪婪算法。他们首先提出两个改进 NEH 启发式生成两个较好初始解，利用局部搜索方法生成其他初始解。然后，为了提高算法性能，提出面向双目标的贪婪搜索策略用于寻找非支配解，引入了面向双目标的接受准则，并采用重启机制避免局部最优。

3）混合流水作业模型

(1) 经济目标

针对 $HF_m,R_m|M_j|\sum T_j$(M_j 表示工件具有机器适用性约束，即工件 j 仅能分配给特定的一些机器，M_j 为工件 j 的适用机器集合)问题，Yu 等(2018)提出了遗传算法，其中设计了一种针对总拖期目标的解码方法获得紧凑的排序。更具体地说，通过分析两种广泛使用的解码方法，即排列排序和列表排序用于交付期相关优化目标时的不足，利用一种基于优先级的机器选择机制避免不必要的机器闲置。Costa 等(2020)研究了人力资源受限(分配至每个生产阶段的工人数量低于同一阶段的机器数量，其中工人负责执行顺序无关的准备操作)的多阶段 $HF\|C_{max}$ 问题，提出基于回溯搜索和禁忌搜索的混合算法，其中设计了多阶段编码结构用于禁忌搜索，采用基于列表调度方法的高效解码算法进行任务调度和人力资源分配，并利用重启机制增强算法的开发能力。针对考

虑人工技能水平、年龄因素、学习和遗忘效应的多阶段混合流水车间调度问题，其中目标函数是极小化最大完工时间和总流经时间的加权和，Marichelvam 等(2020)提出了改进粒子群优化算法来求解该问题，该算法结合 SPT 调度规则和 NEH 启发式进行种群初始化，并引入混合变邻域搜索算法改进粒子群算法产生的近优解。针对缓冲区容量有限的可重入混合流水车间调度，其中目标函数为极小化最大完工时间和平均流经时间加权和，Lin 等(2020)开发了一种基于和声搜索和遗传的混合算法进行求解，其利用遗传算法的交叉和变异算子增强局部寻优能力，利用和声搜索增强全局寻优能力，增强解的多样性。针对异速柔性流水作业问题，Aqil 等(2020)提出改进迭代贪婪与局部搜索两种算法，其中基于 NEH 启发式初始化的迭代贪婪算法在求解质量和收敛速度上有较好的效果。Lin 等(2021)针对多阶段 HF $\|$ $C_{\max}$ 问题提出了混沌增强模拟退火算法，其设计了混合解码机制和有效的邻域搜索机制来提高解质量，利用基于混沌的退火机制避免陷入局部最优，并通过使用混沌数来调整温度，有助于算法的快速收敛。Martins 等(2021)研究了柔性流水作业与车辆路径规划的联合问题，提出有偏随机迭代变量邻域下降算法，其中改进了编码策略、初始化方法和邻域生成算子。针对异速处理机带有调整时间的情况，Aqil 等(2021)提出了水波优化算法，其中采用了贪婪随机自适应搜索、路径链接技术等改进策略。针对考虑退化效应且缓存受限的双目标问题，Zheng 等(2021)提出了基于遗传、模拟退火和变邻域搜索算法的混合算法，通过对遗传算法变异算子和交叉算子的改进，较好地平衡了全局搜索和局部寻优。针对极小化最大完工时间与平均拖期问题，Gheisariha 等(2021)提出了多目标和声搜索算法进行优化，其中利用构造启发式算法生成初始解，并采用聚类算法进行进一步优化。

(2) 绿色目标

针对考虑目标相对重要性和能效的 HF $\|$ $(\sum T_j, C_{\max}, \mathrm{TEC})$ 问题(总能耗目标的重要性低于其他目标)，Li 等(2019)提出了提出一种新的帕累托支配定义用于衡量目标相对重要性，采用双层帝国竞争算法进行求解，其中第一层为最强帝国，第二层由其他帝国组成，并设计了新的同化、革命和竞争方式。在所提出的帝国竞争方式中，最强帝国不参与帝国竞争，只有第二层的帝国相互竞争，从而避免早熟，且竞争阶段设置了一个记忆单元，用于避免在获胜帝国中直接加入最弱殖民地。在搜索过程中，在不同的帝国中以不同的同化和革命方式产生新的解，以保持种群的高度多样性，避免陷入局部最优。针对考虑能效的 $\mathrm{HF}_m, R_m \| \mathrm{TEC}$ 问题，Meng 等(2019)提出了一种改进遗传算法，设计了一种新的节能解码方法。针对考虑分时电价的两阶段 $\mathrm{HF}_m \mid p\text{-batch} \mid (C_{\max}, \mathrm{TEC})$ 问题，Wang 等(2020)提出了双目标禁忌搜索算法和双目标蚁群优化算法进行中

大规模问题的求解，其中分别将这两种算法与带有局部搜索策略的构造启发式相结合，并验证了两种算法的性能。Chen 等(2020)研究了考虑能耗的混合流水车间批量调度问题，同时优化最大完工时间和电力能耗两个目标。他们采用NSGA-Ⅱ算法获得近似帕累托解，其中开发了一种多目标能效调度算法用于计算遗传算法中每个染色体的适应度，并利用用户定义的偏好向量控制搜索区域。针对考虑工人柔性和能效的柔性流水车间调度问题，Gong 等(2020)同时优化与绿色生产、处理时间和工人成本相关的目标，并提出一种混合进化算法进行求解。其中设计了有效的初始化方法、编解码算子、交叉算子和变异算子。为了提高算法收敛速度并充分开发算法解空间，设计了一种新的变邻域搜索策略，该策略包含两个邻域搜索算子，可从不同方向移动关键路径上的操作，增强算法的多目标优化能力。针对具有机器相关加工阶段(第一阶段的无关并行机技术水平不同，其中有一些是多功能的，可以对作业执行多个处理过程，即分配给这些机器的作业无须在下一个阶段进行处理)的双目标柔性流水车间调度问题，Hasani 等(2020)同时极小化总能耗和总生产成本，并提出了一种改进NSGA-Ⅱ算法进行求解。针对泛在环境下的多代理绿色混合流水车间动态调度问题，其中优化目标为同时极小化最大完工时间、总能耗和总碳排放量，Shi 等(2021)提出了一种基于遗传算法的可变优先级动态调度优化方法，先利用基于两段编码、指标加权和的遗传算法生成预调度解，再通过基于事件驱动优先级权重的局部搜索策略动态地生成重调度解。Wang 等(2021)同时考虑最大总工期及设备总能耗目标，提出多目标鲸鱼算法求解帕累托前沿，其中给出一种动态编码方法，并改进了初始化策略、掠夺优化方法等组件。Jiang 等(2021)采用基于分解的多目标进化算法同时对总工期和总能耗进行优化，这里设计了根据非支配前沿的分布情况自适应修改权重向量的权重向量动态调整策略。

4) 分布式流水作业模型

(1) 经济目标

Li 等(2019)研究了考虑退化效应的 $DF_m, R_m \mid p\text{-batch}, p_{ij}^A = p_{ij} + \beta t \mid C_{\max}$($p_{ij}^A = p_{ij} + \beta t$ 是考虑退化效应的工件实际处理时间，p_{ij} 和 t 分别是工件 j 在机器 i 上的正常处理时间、开始时间，β 是所有工件的共同恶化率)问题，并提出混合人工蜂群算法进行求解。首先基于批处理定理分别提出批次分配启发式和右移启发式，用于将作业划分到合适的批次中并进行调度。然后，针对分布式流水车间和平行批处理段设计了 5 种局部搜索算子，提高了算法的开发能力。此外，基于全局和局部最优解的相关信息提出了一种新型侦察蜂启发式，显著地提升了搜索性能。针对 $DF_m \mid prmu \mid C_{\max}$ 问题，Ruiz 等(2019)提出了迭代贪婪算法，设计了初始化方法，并提出有偏破坏算子(用于删除 $C_{\max}$ 最

大工厂 50%的工件)和局部强化搜索方法。针对 $DF_m|prmu,ST_{sd}|C_{max}$ 问题，Huang 等(2020)提出了具有重启机制的迭代贪婪算法，他们通过分析问题性质提出了两种基于任务块移动的邻域结构用于局部搜索，其中第一种在关键工厂上操作，另一种在每个单独的工厂中使用。然后，基于问题性质舍弃了传统迭代贪婪中常用的类模拟退火接收准则，提出了一种具有 6 种邻域算子的重启方法，保证了解的多样性，有助于算法跳出局部最优。此外，在迭代贪婪搜索阶段，利用控制参数实现了算法开发和探索的平衡。针对具有有限缓冲区的 $DF_m \parallel C_{max}$ 问题，Zhang 等(2019)分别提出了连续差分进化算法和离散差分进化算法进行求解，其中提出了两种构造启发式(分布式 SPT 和分布式 NEH)用于快速初始化。为了增强算法的局部搜索能力，设计基于关键工厂的邻域算子用于改进种群最优个体和随机个体，最后验证了离散差分进化算法的性能更优。针对具有交付期窗口的 $DF|prmu|TWET^{dw}$(目标函数为极小化具有交付期窗口的提前和拖期加权和“total weighted earliness and tardiness with due windows”)问题，Jing 等(2020)提出了带有空闲时间插入评价指标的迭代贪婪算法，其设计了基于单位提前和拖期权重、EDD 规则的 NEH 启发式用于种群初始化，并验证其有效性。然后，引入空闲时间插入思想使每个作业的完成时间在到期窗口内或尽可能接近到期窗口，并基于此思想设计了具有动态大小的破坏和重构策略和局部搜索方法，还给出了解的选择公式。针对 $DF_m|block|C_{max}$ 问题，Shao 等(2020a)提出了混合离散果蝇优化算法，其中根据问题特征重新设计了嗅觉觅食和视觉觅食算子。在嗅觉觅食阶段中，提出了一种新的基于插入的邻域算子用于全局搜索；在视觉觅食阶段中，引入了一种模拟退火接受标准来帮助算法跳出局部最优。之后，针对分布式阻塞流水车间模糊调度问题(处理时间不确定，用模糊数表示)，其中优化目标为极小化所有工厂的模糊最大完工时间，Shao 等(2020b)首先基于问题特定知识和 NEH 分别提出了两个构造启发式，即 INEH 和 DPFNEH；INEH 利用模糊处理时间的扩展值来生成初始作业序列，DPFNEH 通过减少期望空闲时间总和及阻塞时间将部分任务分配给工厂。然后，提出了两种迭代贪婪算法，利用以上两个启发式产生高质量的初始解，引入基于高原探索的局部搜索来提高解的质量，并设计了基于模糊特征的改进接受准则，避免陷入局部最优。针对 $DF_m|prmu,block|C_{max}$ 问题，Zhao 等(2020c)提出了集成离散差分进化算法，该算法集成了两种启发式方法和一种随机策略，为分布式环境提供了一组理想的初始解，其中启发式方法考虑了前端延迟、阻塞时间和空闲时间。然后，重新设计变异、交叉和选择算子，辅助所提出算法在离散域中执行；此外，还在算法框架中引入了精英保留策略，平衡算法的开发和探索能力。针对具有预防性维护操作的 $DF_m|prmu|C_{max}$ 问题，

Mao 等(2021)提出了多起点迭代贪婪算法,该算法在 NEH2 中增加了一个删除操作生成高质量和高分散性的解,用于初始化和重新初始化。此外,提出了 3 种有效算子加强局部邻域解的开发,并设计了带有锦标赛选择方法的破坏策略和引入增强策略的重构策略避免陷入局部最优。针对具有批量流和结转顺序相关准备时间的异构 $DF_m|prmu,ST_{csd}|C_{max}$($ST_{csd}$ 表示结转序列相关准备时间"carryover sequence-dependent setup time",其依赖于待加工的作业和之前所有已加工完成的作业,而不仅是前一个作业)问题,Meng 等(2021)提出了改进人工蜂群算法,该算法利用改进 NEH2 启发式方法生成初始种群,在跟随蜂阶段设计了一种个体间的合作机制用于促进种群个体间优化信息的传递,并在侦察蜂阶段设计了种群重启策略。

(2) 绿色目标

针对考虑能效的 $DF_m|prmu,no\text{-}idle|(C_{max},TEC)$ 问题,Chen 等(2019)总结了问题性质,并提出多目标协同优化算法进行求解,其中采用双启发式协同初始化方法保证种群的质量和多样性,设计了多个适用于双目标优化的顺序相关和速度相关邻域算子,分别针对非支配和被支配个体设计了不同的局部搜索强化策略,并对非关键操作设计速度调整策略改进总能耗目标。针对 $DF_m|prmu,ST_{si}|(C_{max},TEC)$($ST_{si}$ 表示顺序不相关准备时间"sequence-independent set-up time")问题,Wang 等(2020)提出了基于自适应能效算子的多目标鲸鱼群算法,其编码包括作业排列、工厂分配和处理速度 3 个部分;为了提高解的质量,他们还设计了问题相关的局部搜索和更新开发机制,分别对最大完工时间和总能耗进行优化,并引入 NSGA-II 算法的非支配排序进行种群更新。针对考虑能效和异构工厂(工厂设置为置换流水车间或混合流水车间)的 $DF\|(C_{max},TEC)$ 问题,Lu 等(2020)提出了混合多目标迭代贪婪算法,其嵌入了一种新的节能策略降低能耗和预防机器故障,并提出了一种协同启发式生成初始种群,不仅产生了 3 个高质量的初始解,而且能保证种群的多样性;此外,他们还基于问题性质设计了基于关键工厂和空闲时间间隔的局部搜索策略,提高了开发能力。针对考虑能效的分布式置换流水车间逆向调度问题,其中目标函数为同时极小化处理时间变化总和与总能耗,Mou 等(2022)提出了引入协同搜索机制的混合遗传算法,该算法利用改进 NEH 启发式方法和调度规则进行初始化,设计了符合问题特征的交叉和变异机制,并采用了基于双模局部搜索策略和学习机制的双种群协同搜索方法。

5) 分布式装配流水作业模型

针对 $DAF_m|prmu|C_{max}$(DAF 表示分布式装配流水车间"distributed assembly flow shop")问题,Pan 等(2019)提出了迭代贪婪算法,该算法基于

NEH 启发式生成初始解，使用变邻域搜索作为局域搜索方法，并提出了一些插入邻域加速计算公式。针对 $DAF_m|prmu|C_{max}$ 问题，Ferone 等(2020)提出了有偏随机迭代局部搜索算法，该算法引入基于蒙特卡洛模拟的有偏随机技术构建初始解，并在此基础上应用迭代局域搜索算法探索可行解空间。针对 $DAF_m|prmu|\sum C_j$ 问题，Huang 等(2021)提出了一种基于分组思想的改进迭代贪婪算法，利用分组思想不仅增加了解的多样性，而且扩大了搜索空间。该算法采用基于 Palmer 启发式的任务分配规则和基于 NEH 启发式的产品分配规则用于初始化。然后，为了提升算法的效率，基于两阶段算子针对产品和任务分别设计了破坏重构和局部搜索策略，其中在破坏阶段根据实例规模自适应地选择作业，并设计了考虑种群个体目标值和迭代次数的选择方法。针对两阶段 $DAF_m \parallel C_{max}$，Lei 等(2021)提出了教学协同优化算法。该算法将种群划分为多个班级增强了种群多样性，其整个搜索过程由两个阶段组成：所有班级在第一阶段独立进化，在第二阶段协同进化。每个阶段中设计了两个教学阶段和一个学习阶段并依次执行，充分利用了最优个体的潜在优势改善种群质量。第二阶段是最优班级和最差班级间进行交互学习的过程，其基于差异化搜索策略控制最差班级的搜索次数转移至最优班级，最优班级的搜索能力转移至最差班级，从而避免在最差班级上浪费计算资源，这种策略平衡了算法的探索和开发能力。

6) 分布式混合流水作业模型

针对带多处理器任务的 $DHF \parallel C_{max}$ 问题，Cai 等(2020)提出了动态混合蛙跳算法，采用双字符串编码和基于多处理机任务性质的解码方法。该算法结合全局搜索和动态多邻域搜索算子，针对至少可改进两个解的族群进行动态搜索，避免计算资源的浪费，并根据邻域结构的历史优化效果对其进行动态选择。此外，针对族群最优解设计了破坏与重构方法，增强算法的局部搜索能力。之后，Cai 等(2021)研究了 $DHF|ST_{sd}|\left(C_{max},\sum T_j\right)$ 问题，通过集成第一阶段的工厂和机器分配降低子问题的耦合程度，并提出考虑族群质量的混合蛙跳算法进行求解。该算法重新定义了族群解的评价指标，并设计基于族群解质量的新型动态搜索和基于族群进化质量的动态族群重构方法，平衡了算法的开发和探索能力。针对 $DHF \parallel C_{max}$ 问题，Shao 等(2020)设计了多邻域迭代贪婪算法，其利用多邻域算子来增强局部搜索能力，并结合变邻域下降框架来改进重构解。之后，针对 $DHF|nwt|C_{max}$ 问题，Shao(2021)设计了一种变邻域下降算法进行求解，其中提出了 3 个基于 NEH 的工厂分配规则和 14 条调度规则用于生成初始解，并根据分布式流水车间的特征设计了 5 种邻域搜索算子。针对 $DHF|block,prmu|C_{max}$ 问题，Shao(2021)提出了一种迭代贪婪算法，该算法利用改进 NEH 启发式产生初始解，设计基于问题特定知识的破坏-重建策

略进行探索和扰动，并提出 3 个局部搜索策略改进候选解。考虑到分布式制造系统中的不确定性，Zheng 等(2020)研究了具有模糊加工时间和模糊交付期的多目标模糊分布式混合流水车间调度问题，其中优化目标为同时极小化模糊总拖期和极大化鲁棒性，他们通过结合分布估计和迭代贪婪搜索算法，提出了一种具有问题特定策略的协同进化算法。该算法利用带有虚拟工件的特定编解码方法处理车间分配、作业排序和机器分配的强耦合关系。在分布估计模式搜索中，建立了问题特定的概率模型以缩小搜索空间，并提出了一种采样机制产生新个体。为了增强开发能力，他们设计了一个基于关键车间的局部搜索来提高非支配解的性能。在迭代贪婪模式搜索中，采用了破坏和重建方法，以进一步开发更好的解决方案。此外，为了平衡探索和开发能力，他们基于信息熵和精英解的多样性，设计了一种模式切换的协作方法。

针对异构 DHF，$R_m|\mathrm{ST}_{\mathrm{sd}}|C_{\max}$ 问题，Li 等(2020)提出一种改进的人工蜂群算法，该算法采用双层编码和基于机器选择的解码方法保证解的可行性，并采用工厂分配规则和结合 NEH 的迭代贪婪规则生成高质量的初始解。他们还提出了新的解更新技术，包括基于关键工厂的局部开发策略，结合模拟退火和保留机制优点的混合搜索策略，这些技术可以保持解空间的多样性，提高计算效率。之后，针对 $\mathrm{DHF}_m|\mathrm{ST}_{\mathrm{sd}}|C_{\max}$ 问题，Li 等(2021)提出了离散人工蜂群算法。该算法采用双层编码和结合最早可用机器和最早完工时间规则的解码方法，并提出平均工厂分配策略和两种 NEH 算法用于生成初始种群，其中平均工厂分配策略可确保每个工厂至少被分配一项工作。

1.5 本书主要内容

本书是作者研究团队近年来代表性研究成果的总结，重点讨论如何应用分支定界及智能优化算法求解流水作业调度问题，同时也对某些基于调度规则的启发式进行了渐近性能分析。全书共分为 6 章，主要内容概述如下：第 1 章为绪论，简要介绍调度问题的描述与求解算法，算法性能分析方法；第 2 章介绍带有释放时间的流水作业调度问题，其中考虑了非线性目标问题；第 3 章介绍了考虑处理器阻塞的流水作业调度问题，其中研究了与客户满意相关的目标函数；第 4 章介绍了考虑学习效应的流水作业调度问题，其中证明了基于最短处理时间优先启发式的渐近最优性；第 5 章、第 6 章分别介绍了双代理流水作业调度问题和双代理阻塞流水作业及其扩展问题，其中研究了一类基于优势代理优先启发式的理论性能；第 7 章介绍了考虑学习效应的混合流水作业调度问题。

第 2 章　带有释放时间的流水作业调度问题

2.1　引言

流水作业调度是制造业中最常见的优化模型。例如：用于治疗冠心病的冠脉支架的生产过程，首先，使用激光切割机对钛合金管进行激光雕刻；其次，将半成品浸入酸性溶液去除表面氧化层；再次，经过机械研磨去除支架表面金属毛刺；最后，采用电化学抛光工艺增强支架表面的光亮度。显然，上述过程可以抽象为流水作业调度模型。企业的优化目标可能包括降低能耗、减小在制品库存或提高客户满意度等。

本章讨论带有释放时间的流水作业调度模型，其中目标函数分别为最大完工时间、最大送达时间与完工时间 $k(k\geqslant 2)$ 次方和，利用三参数表示法，可以分别表示为 $F_m|r_j|C_{\max}$、$F_m|r_j|Q_{\max}$ 和 $F_m|r_j|\sum C_j^k$（分别简记为问题 2.1、问题 2.2 和问题 2.3）。从工业意义上讲，最大完工时间指标能够有效减少机器负载进而降低能耗，是实现绿色制造的关键目标；最大送达时间指标综合考虑了生产与物流过程，是提高客户满意度的重要目标；完工时间 k 次方和指标能够同时降低生产能耗与在制品库存，避免双目标线性化之后与原问题之间的偏差。为了便于研究，通常假设流水作业调度模型中所有待处理任务同时可用，但这种情况是非常理想的状态，在实际调度过程中，任务是陆续释放到系统中。因此，考虑带有释放时间的流水作业调度模型更符合实际情况。

关于流水作业调度的研究工作主要集中于任务同时可用的情况，而任务带有不同释放时间的情况，相关文献并不多见。Lenstra (1977)证明了 $F_2|r_j|C_{\max}$ 是强 NP-难问题，这说明处理器数多于两台的一般性问题无法在多项式时间内求得最优解，除非 P=NP。Potts (1985)提出若干多项式时间算法近似求解 $F_2|r_j|C_{\max}$ 问题。针对 $F_2|r_j|C_{\max}$ 和 $F_3|r_j|C_{\max}$ 问题，Tadei 等(1998)和 Cheng 等(2001)分别设计了分支定界(branch & bound，B&B)算法进行最优求解。但是，因为该算法中采用的优势规则与节点下界是为两台和三台处理器模型而特别设计，所以无法将其扩展用于求解 $F_m|r_j|C_{\max}$ 问题。Bai 等(2010)研究了 $F_m|r_j|C_{\max}$ 问题，证明了先来先做(first-come，first-served)规则具有渐近最优性。Zabihzadeh 等(2016)采用遗传算法和蚁群优化算法求解

$FF_m|r_j$,block$|C_{\max}$ 问题。Kaminsky 等(2003)指出 $F_m|r_j|Q_{\max}$ 为 NP-难问题,并在概率极限意义下证明了 LDT 启发式具有渐近最优性。考虑到 $d_j=-q_j$,即最大延误与最大送达时间有等价性,因此,这里提及有关最大延误目标的结论。针对 $F_m|r_j|Q_{\max}$ 问题,Grabowski 等(1983)利用分支定界算法进行最优求解。但是,该分支定界只是一个普通的算法框架,并没有提出有效的剪支策略。

流水作业完工时间 k 次方和问题的研究工作主要集中于线性目标函数($k=1$)的情况,而有关非线性目标函数($k\geqslant 2$)的成果较少。Koulamas 等(2005)指出 $F_m\|\sum C_j^2$ 是 NP-难问题,并分析了最短处理时间优先(shortest processing time first,SPT)启发式的渐近最优性与最坏性能比。Xu 等(2008)利用 SPT 启发式求解考虑学习效应的流水作业极小化完工时间平方和问题,并对该算法进行了最坏性能分析,这里学习效应分别为线性函数与幂函数。Wang 等(2012)将 SPT 启发式的最坏情况比进一步推广至指数学习效应函数的情况。Bai(2015)指出 $F_m|r_j|\sum C_j{}^2$ 是 NP-难问题,并证明了可用最短处理时间(shortest processing time availble,SPTA)启发式具有渐近最优性。Bai 等(2014)将基于 SPT 和 SPTA 两类启发式算法的渐近最优性分别推广至 $F_m\|\sum C_j{}^k$ 和 $F_m|r_j|\sum C_j{}^k$ 问题。针对 $F_m\|\sum w_jC_j{}^2$ 问题,Ren 等(2017)分析了加权最短处理时间(weighted shortest processing time,WSPT)启发式的渐近最优性与最坏性能比,并利用离散差分进化(discrete differential evolution,DDE)算法求得近优解。

针对问题 2.1、问题 2.2 和问题 2.3,本章建立了混合整数规划(mixed integer programming,MIP)模型,用于商业优化软件求解;设计了分支定界算法进行最优求解小规模问题;采用离散差分进化算法近似求解中等规模问题;通过数值仿真验证这些算法的有效性。

2.2 问题描述与数学模型

在流水作业调度模型中,n 项不同的任务按照相同的工艺路线经过 m 台串联处理器加工。工序 $O_{i,j}$ 表示任务 j ($j=1,2,\cdots,n$)需要经过处理器 i ($i=1,2,\cdots,m$)执行。处理时间 $p_{i,j}\geqslant 0$ 表示执行工序 $O_{i,j}$ 花费的时间。任务 j 在释放时间 r_j 进入系统,这是该任务可以开始执行的最早时刻。运送时间 q_j 表示任务 j 完成处理之后交付给客户的时间。完工时间 $C_{i,j}$ 表示工序 $O_{i,j}$ 结束加工的时刻,$C_{m,j}$ 简记为 C_j。送达时间 $Q_j=C_j+q_j$,表示任务 j 交付给客户的时刻。每台处理器按照 FCFS 规则执行所有任务,即同顺序处理模式:这些

任务按照相同的顺序经过每台处理器。任意两台相邻处理器之间的缓存能力是无限的。每项任务在处理过程中不允许中断，即某项任务一旦开始处理就要持续至其完工为止。在相同时刻，一台处理器只能执行一项任务，而且一项任务只能由一台处理器加工。优化目标分别为极小化最大完工时间 $C_{\max}=\max\{C_1,C_2,\cdots,C_n\}$，最大送达时间 $Q_{\max}=\max\{Q_1,Q_2,\cdots,Q_n\}$，与完工时间 k 次方和 $\sum C_j^k=\min\sum_{j=1}^{n}(C_j)^k$。本章的 MIP 模型是后续更加复杂的流水作业模型的基础。

为了建立 MIP 模型，给出下面一系列相关符号：

$S_{i,j}$：工序 $O_{i,j}$ 的开始时间，$i=1,2,\cdots,m$；$j=1,2,\cdots,n$；

$x_{j,j'}$：0-1 变量。若任务 j 为任务 j' 的紧后任务（紧接在该任务之后的任务），则为 1，否则为 0，$j\neq j'$；

θ：足够大的正数。

根据以上各定义，将上述问题表示为数学规划模型：

$$\text{minimize } F(C_j),F(C_j)\in\left\{C_{\max},Q_{\max},\sum C_j^k(k\geqslant 2)\right\}$$

$$\text{s.t.}\sum_{j=0}^{n}x_{j,j'}=1,\ j'=0,1,\cdots,n \tag{2-1}$$

$$\sum_{j'=0}^{n}x_{j,j'}=1,\ j=0,1,\cdots,n \tag{2-2}$$

$$\sum_{j=0}^{n}x_{j,j}=0,\ j=0,1,\cdots,n \tag{2-3}$$

$$x_{j,j'}+x_{j',j}\leqslant 1,\ j,j'=0,1,\cdots,n \tag{2-4}$$

$$S_{i,j}-r_j\geqslant 0,\ i=1,2,\cdots,m;\ j=1,2,\cdots,n \tag{2-5}$$

$$S_{i,j}+p_{i,j}=C_{i,j},\ i=1,2,\cdots,m;\ j=1,2,\cdots,n \tag{2-6}$$

$$C_{i+1,j}-C_{i,j}-p_{i+1,j}\geqslant 0,\ i=1,2,\cdots,m-1;\ j=1,2,\cdots,n \tag{2-7}$$

$$C_{i,j}-C_{i,j'}\geqslant p_{i,j}-\theta(1-x_{j,j'}),\ i=1,2,\cdots,m;\ j,j'=1,2,\cdots,n \tag{2-8}$$

$$x_{j,j'}\in\{0,1\},r_j\geqslant 0,p_{i,j}\geqslant 0,C_{i,j}\geqslant 0,\ i=1,2,\cdots,m;\ j=1,2,\cdots,n \tag{2-9}$$

约束(2-1)～约束(2-4)限制了每项任务只能有一项紧前任务（紧接在该任务之前的任务），且每项任务也只能有一项紧后任务。为了保证约束的完整性，模型中假设存在一个虚拟任务 $j=0$，在所有处理器上的处理时间均为 0。该任务是第一项任务的紧前任务，是最后一项任务的紧后任务。约束(2-5)表示任

务的开始时间不得早于其释放时间;约束(2-6)定义了任务在同一台处理器上的执行过程;约束(2-7)限定同一任务在相邻处理器之间的执行顺序;约束(2-8)描述相邻任务在同一台处理器上执行顺序。约束(2-9)定义了相关变量及参数的取值范围。

2.3 分支定界算法

分支定界是一种用于求解 NP-难问题的隐枚举算法框架,通过系统地搜索状态空间获得小规模问题的最优解。在搜索过程中,深度优先与广度优先技术是常用的两种重要分支策略。前者在回溯之前尽可能沿着搜索树的每个分支进行探索;后者首先探索当前深度的所有相邻节点,然后再探索下一个深度的节点。考虑到本书讨论的是同顺序流水作业调度模型,因此应用深度优先加回溯技术可以剪掉大量节点,节约计算资源。分支定界算法从搜索树的虚拟根节点(空任务序列)开始,沿着搜索树分支搜索每个候选节点,其中每个节点(根节点和叶子节点除外)表示一个给定的部分任务序列。虽然一般的分支定界算法不需要任何加速技术就能得到最优解,但是求解过程却要花费相当长的时间。为了提升分支定界算法的求解效率,常用的加速策略是设计有效的剪支规则和高质量的分支定界算法下界。本章基于任务释放时间的特性及单机最优调度规则,分别提出了剪支规则与分支定界算法下界来减少有效节点数量,进而提高算法求解速度。

2.3.1 剪支规则

考虑带有释放时间的流水作业调度模型中某个可行调度,若将某项已经释放的任务延迟至较晚释放任务完工之后开始处理,则必会人为地产生较多空闲时间,导致目标函数值恶化。这种情况可以归纳为如下性质。

性质 2.1 对于某个已经固定 $j-1$ 项任务顺序的部分序列,若其紧后任务 j 满足条件:

$$S_{i,j} \geqslant \min_{h \in N'} \{C_{i,h}\} \tag{2-10}$$

则剩余未调度任务的开始时间被延迟,目标函数值恶化。其中,N'表示未调度任务集合,$i=1,2,\cdots,m$,$j=1,2,\cdots,n$。

显然,不等式(2-10)表明,若存在未调度任务 $h \in N'$能够在任务 j 开始之前结束其在所有处理器上的加工,则不必将任务 h 安排在任务 j 之后加工,否则将人为增加处理器空闲时间,导致目标函数值变差。若最后一台处理器不满足条件(2-10),则可以用以下性质替代。

性质 2.2　针对问题 2.1，若前 $m-1$ 台处理器满足条件(2-10)，则将未调度任务安排在任务 j 之后处理会导致目标函数值恶化。

证明　最后一台处理器的调度问题可以简化为 $1|r_j|C_{\max}$ 问题，其中任务 j 的释放时间定义为 $\hat{r}_j=\max\{C_{m-1,j},C_{j-1}\}$，$j=1,2,\cdots,n$，已知利用 FCFS 规则可以求得其最优解。条件(2-10)表明有 $\hat{r}_h\leqslant\hat{r}_j$，即在处理器 m 上任务 h 的释放时间早于任务 j，由此可以推出性质 2.2。

性质 2.3　针对问题 2.2，若前 $m-1$ 台处理器满足条件(2-10)且 $q_h\geqslant q_j$，则将未调度任务安排在任务 j 之后处理会导致目标函数值恶化。

证明　最后一台处理器的调度问题可以简化为 $1|r_j|Q_{\max}$ 问题，在满足一致性条件的情况下，即 $q_1\geqslant q_2\geqslant\cdots\geqslant q_n$ 且 $r_1\leqslant r_2\leqslant\cdots\leqslant r_n$，根据 LDT 规则可以求得最优解。注意到 $\hat{r}_h\leqslant\hat{r}_j$ 且 $q_h\geqslant q_j$，即可推出性质 2.3。

性质 2.4　针对问题 2.3，若前 $m-1$ 台处理器满足条件(2-10)且 $p_{m,h}\leqslant p_{m,j}$，则将未调度任务安排在任务 j 之后处理会导致目标函数值恶化。

证明　最后一台处理器的调度问题可以简化为 $1|r_j|\sum C_j^k$ 问题，在满足一致性条件的情况下，即 $p_1\leqslant p_2\leqslant\cdots\leqslant p_n$ 且 $r_1\leqslant r_2\leqslant\cdots\leqslant r_n$，根据 SPT 规则可以求得最优解。注意到 $\hat{r}_h\leqslant\hat{r}_j$ 且 $p_{m,h}\leqslant p_{m,j}$，即可推出性质 2.4。

在分支过程中，将性质 2.1 分别与性质 2.2～性质 2.4 组合后，形成剪支规则 2.1～规则 2.3 用于求解问题 2.1、问题 2.2 和问题 2.3。针对搜索树上的每个候选节点，利用上述剪支规则判断是否需要进行分支。剪支规则剪掉的无效节点越多，分支定界算法的搜索空间就越小，求解效率越高。

2.3.2　分支定界算法下界

针对给定搜索树上每个有效的分支节点，分支定界算法下界的主要作用就是估计其中未调度任务可能的最好目标值。考虑目标函数为极小化的调度问题，每当求得某分支的上界且更新了当前最好上界之后，就要回溯剪支，删除那些下界值不小于当前最优值的节点。显然，分支定界算法下界越有效、越接近最优解，则剪掉的节点数越多。因此，充分利用给定问题的输入信息，设计高效的分支定界算法下界是提升分支定界求解效率的途径之一。

考虑某分支节点 $\pi(j)=([1],[2],\cdots,[j])$，$j\in N$，表示其中前 j 项任务已经做好安排，这里 N 表示系统中所有任务的集合，$[j]$表示位于第 j 位置的任务。不失一般性，假设包含未调度任务的部分序列可表示为 $\pi'(j+1)=([j+1],[j+2],\cdots,[n])$。工序 $O_{i,[h]}$的最早可用时间 $R_{i,[h]}$可以表示为如下形式，这里 $h\in N'$。

在第一台处理器上：

$$R_{1,[h]}=\max\{C_{1,[j]},r_h\}$$

在第 i 台处理器上：

$$R_{i,[h]}=\max\{C_{i,[j]},R_{i-1,h}+p_{i-1,h}\}$$

式中，$i=2,3,\cdots,m$。显然，若不考虑任务前后处理关系的约束，则求每台处理器上未调度工序的最优解问题可简化为带有释放时间的单机调度问题。在处理器 i 上，工序 $O_{i,[h]}$ 完工时间的可能估计值为

$$C_{[h]}\geqslant\max_{j+1\leqslant x\leqslant h}\left\{R_{i,[x]}+\sum_{u=x}^{h}p_{i,[u]}\right\}+\sum_{i'=i+1}^{m}p_{i',[h]} \tag{2-11}$$

在不等式(2-11)右端项中，除 $\sum_{u=x}^{h}p_{i,[u]}$ 与顺序相关外，其余项都是常数。根据 FCFS 规则可求得 $1|r_j|C_{\max}$ 问题最优解，容易得到分支定界求解问题 2.1 的分支定界算法下界 LB2.1，其目标值表示为

$$Z_{2.1}^{\mathrm{LB}}=\max_{1\leqslant i\leqslant m}\left\{\max_{j+1\leqslant x\leqslant n}\left\{R_{i,[x]}+\sum_{h=x}^{n}p_{i,[h]}^{\mathrm{FCFS}}\right\}+\min_{h'\in N'}\sum_{i'=i+1}^{m}p_{i',[h']}\right\}$$

式中，$p_{i,[h]}^{\mathrm{FCFS}}$ 表示工序 $O_{i,[h]}$ 按照 FCFS 规则进行调度。

根据可中断 LDT(PLDT)规则可求得 $1|r_j,\mathrm{prmp}|Q_{\max}$ 问题最优解，参考不等式(2-11)能够求得工序 $O_{i,[h]}$ 最大送达时间的可能估计值为

$$Q_{[h]}^{\mathrm{LB}}=\max_{1\leqslant i\leqslant m}\left\{\max_{j+1\leqslant x_1\leqslant x_2\leqslant h}\left\{R_{i,[x_1]}+\sum_{u=x_1}^{x_2}p_{i,[u]}^{\mathrm{PLDT}}+q_{[x_2]}\right\}+\min_{u'\in N'}\sum_{i'=i+1}^{m}p_{i',[u']}\right\}$$

式中，$p_{i,[h]}^{\mathrm{PLDT}}$ 表示工序 $O_{i,[h]}$ 按照 PLDT 规则进行调度。根据最大送达时间的定义，容易得到分支定界求解问题 2.2 的分支定界算法下界 LB2.2，其目标值表示为

$$Z_{2.2}^{\mathrm{LB}}=\max\{\max_{1\leqslant h_1\leqslant j}Q_{[h_1]},\max_{j+1\leqslant h_2\leqslant n}Q_{[h_2]}^{\mathrm{LB}}\}$$

根据最短剩余处理时间优先(shortest remaining processing time first，SRPT)规则可求得 $1|r_j,\mathrm{prmp}|\sum C_j{}^k$ 问题最优解(Bai et al，2014)，参考不等式(2-11)能够求得工序 $O_{i,[h]}$ 完工时间的可能估计值为

$$C_{[h]}^{\mathrm{LB}}=\max_{j+1\leqslant x\leqslant h}\left\{R_{i,[x]}+\sum_{u=x}^{h}p_{i,[u]}^{\mathrm{SRPT}}\right\}+\sum_{i'=i+1}^{m}p_{i',[h]} \tag{2-12}$$

式中，$p_{i,[u]}^{\mathrm{SRPT}}$ 表示工序 $O_{i,[h]}$ 按照 SRPT 规则进行调度。搜索树上的每个有效节点都包含已安排任务和未安排任务，于是目标函数可以表示为

$$\sum_{j=1}^{n}C_j^k=\sum_{h_1=1}^{j}C_{[h_1]}^k+\sum_{h_2=j+1}^{n}C_{[h_2]}^k \tag{2-13}$$

等式(2-13)右边第二项是未排任务的完工时间 k 次方和，将式(2-12)代入有

$$\sum_{h_2=j+1}^{n} C_{h_2}^{k} \geqslant \max_{1\leqslant i\leqslant m}\left\{\sum_{h_2=j+1}^{n}(C_{[h_2]}^{\mathrm{LB}})^{k}\right\}$$

$$=\max_{1\leqslant i\leqslant m}\left\{\sum_{h_2=j+1}^{n}\left(\max_{j+1\leqslant x\leqslant h_2}\left\{R_{i,[x]}+\sum_{u=x}^{h_2}p_{i,[u]}^{\mathrm{SRPT}}\right\}+\sum_{i'=i+1}^{m}p_{i',[h_2]}\right)^{k}\right\}$$

令 $\max\limits_{j+1\leqslant x\leqslant h_2}\left\{R_{i,[x]}+\sum\limits_{u=x}^{h_2}p_{i,[u]}^{\mathrm{SRPT}}\right\}=C_{i,[h_2]}^{\mathrm{SRPT}}$，则有

$$\sum_{h_2=j+1}^{n} C_{h_2}^{k}$$

$$=\max_{1\leqslant i\leqslant m}\left\{\sum_{h=j+1}^{n}\left(C_{i,[h_2]}^{\mathrm{SRPT}}+\sum_{i'=i+1}^{m}p_{i',h_2}\right)^{k}\right\}$$

$$=\max_{1\leqslant i\leqslant m}\left\{\sum_{h=j+1}^{n}(C_{i,[h_2]}^{\mathrm{SRPT}})^{k}+\sum_{u=1}^{k-1}\prod_{\alpha=1}^{u}\frac{k-(\alpha-1)}{\alpha}(C_{i,[h_2]}^{\mathrm{SRPT}})^{k-u}\sum_{i'=i+1}^{m}(p_{i',h_2})^{u}+\sum_{i'=i+1}^{m}(p_{i',h_2})^{k}\right\}$$

$$\geqslant\max_{1\leqslant i\leqslant m}\left\{\sum_{h_2\in N'}(C_{i,[h_2]}^{\mathrm{SRPT}})^{k}+\sum_{u=1}^{k-1}\prod_{\alpha=1}^{u}\frac{k-(\alpha-1)}{\alpha}\left(\min_{h_2\in N'}\left\{\sum_{i'=i+1}^{m}p_{i',[h_2]}\right\}\right)^{u}\times\sum_{h_2\in N'}(C_{i,[h_2]}^{\mathrm{SRPT}})^{k-u}+\sum_{h_2\in N'}\left(\sum_{i'=i+1}^{m}(p_{i',[h_2]})\right)^{k}\right\}\tag{2-14}$$

将式(2-14)代入式(2-13)，即可得到分支定界求解问题 2.3 的分支定界算法下界 LB2.3，其目标值表示为

$$Z_{2.3}^{\mathrm{LB}}=\sum_{h_1=1}^{j}C_{h_1}^{k}+\max_{1\leqslant i\leqslant m}\left\{\sum_{h_2\in N'}(C_{i,[h_2]}^{\mathrm{SRPT}})^{k}+\sum_{u=1}^{k-1}\prod_{\alpha=1}^{u}\frac{k-(\alpha-1)}{\alpha}\left(\min_{h_2\in N'}\left\{\sum_{i'=i+1}^{m}p_{i',[h_2]}\right\}\right)^{u}\times\sum_{h_2\in N'}(C_{i,[h_2]}^{\mathrm{SRPT}})^{k-u}+\sum_{h_2\in N'}\left(\sum_{i'=i+1}^{m}(p_{i',[h_2]})\right)^{k}\right\}$$

式中，当 $i=m$ 时 $\sum\limits_{i'=i+1}^{m}p_{i',[h_2]}=0$。

2.3.3 算法流程

通常，采用调度规则生成分支定界算法的初始上界。令 $P_j=\frac{1}{m}\sum_{i=1}^{m}p_{i,j}$，$j\in N$。针对问题 2.1，采用可用 LPT 优先(LPTA)启发式生成初始解，即每当首台处理器出现空闲，在当前可用任务中优先安排 P_j 值最大的任务。针对问题 2.2，采用可用 LDT 优先(LDTA)启发式生成初始解，即每当首台处理器出现空闲，在当前可用任务中优先安排运送时间最长的任务。针对问题 2.3，采用可用 SPT 优先(SPTA)启发式生成初始解，即每当首台处理器出现空闲，在当前可用任务中优先安排 P_j 值最小的任务。在给定的搜索树上，分支定界算法沿着每个分支从根节点搜索到叶子节点，在此处能够获得新上界。若新上界值优于当前最好上界，则更新后者，并回溯删除无效节点。如果所有节点都被删除，那么当前最好上界就是问题的最优解。否则，重复上述过程，继续搜索具有优势下界值的节点。特别地，如果某节点的下界值等于当前最好上界值，则直接剪掉该节点。执行搜索过程，直至对全部有效节点完成探索。

为了描述算法方便，给出如下符号定义：Z_0^{UB} 表示初始上界值；Z^{UB} 表示当前最好上界值；在第 τ 层搜索树，$Z_{\tau,j}^{\mathrm{LB}}$ 表示位于任务 j 的下界值；在第 $\tau-1$ 层搜索树，$G_{\tau-1,h}$ 表示包括节点 h 全部有效后继节点的集合，其中 $j\in N$，$h\geqslant 0$，$\tau\geqslant 0$。该分支定界算法的主要步骤描述如下。

步骤 1 初始化。对于所有 $j\in N'$，令 $\tau:=0$，$N':=N$，$G_{0,h}=\varnothing$，$Z_{0,j}^{\mathrm{LB}}=0$。采用调度规则生成初始上界 Z_0^{UB}，令 $Z^{\mathrm{UB}}=Z_0^{\mathrm{UB}}$。

步骤 2 分支判断。令 $\tau:=\tau+1$。若 $\tau\leqslant n-2$ 且节点 $j\in N'$ 满足剪支规则，令 $Z_{\tau,j}^{\mathrm{LB}}=\infty$，并将其他节点存入集合 $G_{\tau-1,h}$ 中，转至步骤 3。否则，转至步骤 5。

步骤 3 计算下界。计算每个有效节点 j 的下界值 $Z_{\tau,j}^{\mathrm{LB}}$。对于所有 $j\in G_{\tau-1,h}$，若都满足 $Z_{\tau,j}^{\mathrm{LB}}\geqslant Z^{\mathrm{UB}}$，则转至步骤 7；否则，转至步骤 4。

步骤 4 节点分支。选择满足 $Z_{\tau,\tilde{h}}^{\mathrm{LB}}=\min\limits_{j\in G_{\tau-1,h}}\{Z_{\tau,j}^{\mathrm{LB}}\}$ 的节点 $\tilde{h}$。若存在多个候选节点具有相同的最小下界值，则优先选择释放时间最早的那个节点。返回步骤 2。

步骤 5 更新上界。分别计算节点 $\tilde{h}$ 的两个上界 $Z_{\tilde{h},1}^{\mathrm{UB}}$ 和 $Z_{\tilde{h},2}^{\mathrm{UB}}$。若 $Z^{\mathrm{UB}}>Z_{\tilde{h}}^{\mathrm{UB}}=\min\{Z_{\tilde{h},1}^{\mathrm{UB}},Z_{\tilde{h},2}^{\mathrm{UB}}\}$，则更新 $Z^{\mathrm{UB}}:=Z_{\tilde{h}}^{\mathrm{UB}}$。否则，保持 Z^{UB} 不变，转至步骤 6。

步骤 6 剪支。删除所有满足 $Z_{\tau,j}^{\mathrm{LB}}\geqslant Z^{\mathrm{UB}}$ 的节点，$1\leqslant\tau\leqslant n$ 且 $j\in\bigcup G_{\tau-1,h}$。若 $\bigcup G_{\tau-1,h}=\varnothing$，则转至步骤 7；否则，令 $\tau:=\tau-1$，返回步骤 4。

步骤 7　终止。停止计算，Z^{UB} 即为问题最优解，输出最优解与最优值。

为了便于对分支定界算法求解过程的理解，下面给出数值实例加以说明。

例 2-1　考虑带有释放时间的流水作业调度问题，其中包括 3 台处理器 $\{M_1, M_2, M_3\}$，4 项任务 $\{J_1, J_2, J_3, J_4\}$，目标函数为极小化 $C_{\max}$。任务的释放时间 r_j，处理时间 $p_{i,j}$ 如下所示：

	M_1	M_2	M_3	r_j
J_1	3	8	4	5
J_2	2	4	5	1
J_3	4	5	6	0
J_4	1	7	3	9

使用分支定界算法求解该问题的过程如图 2-1 所示。在第 0 层，采用 LPTA 启发式求得根节点处的初始上界值为 $Z_0^{\mathrm{UB}}=31$。在第 1 层，剪支规则直接剪掉节点 J_1 和 J_4。在第 2 层，节点 J_3 的下界等于 30，然后继续计算上界为 30。于是更新 $Z^{\mathrm{UB}}=30$。至此，所有剩余的有效节点都将被剪掉，最优调度为 $\{J_2, J_3, J_1, J_4\}$。在此例中，只进行了 5 次计算，大大节省了求解时间。

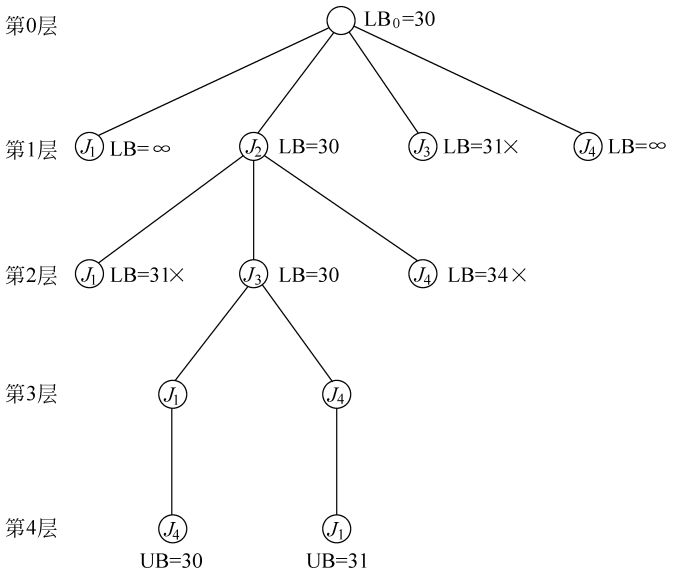

图 2-1　例 2-1 的分支定界搜索树

×表示剪支

例 2-2　考虑带有释放时间的流水作业调度问题，目标函数为极小化 $Q_{\max}$。任务的释放时间 r_j，处理时间 $p_{i,j}$ 与例 2-1 相同，运输时间 q_j 如下所示：

	J_1	J_2	J_3	J_4
q_j	7	3	12	9

使用分支定界算法求解该问题的过程如图 2-2 所示。在第 0 层，采用 LDTA 启发式求得根节点处的初始上界值为 $Z_0^{\mathrm{UB}}=40$。在第 1 层，剪支规则直接剪掉节点 J_1 和 J_4。在第 2 层，节点 J_1 的下界等于 36，继续计算上界，更新 $Z^{\mathrm{UB}}=36$。至此，所有剩余的有效节点都将被剪掉，最优调度为{J_3, J_1, J_4, J_2}。同样，此例中只进行了 5 次计算，大大节省了求解时间。

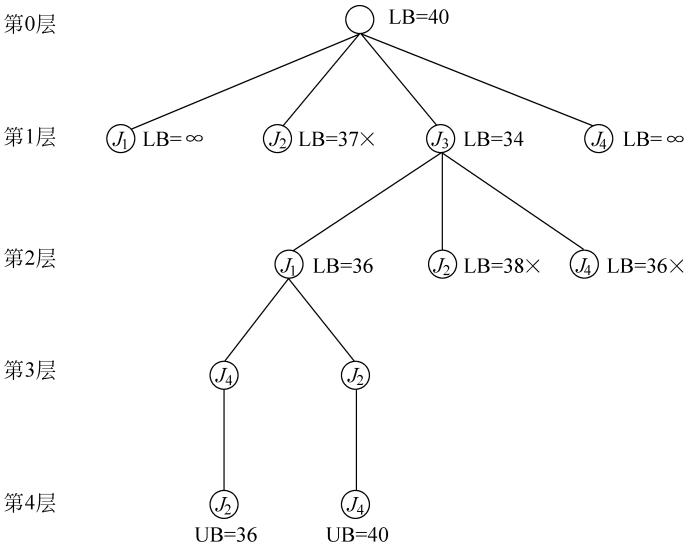

图 2-2 例 2-2 的分支定界搜索树

×表示剪支

例 2-3 考虑带有释放时间的流水作业调度问题，其中包括 3 台处理器{M_1, M_2, M_3}，4 项任务{J_1, J_2, J_3, J_4}，目标函数分别为极小化$\sum C_j{}^2$ 与 $\sum C_j{}^3$。任务的释放时间 r_j，处理时间 $p_{i,j}$ 如下所示：

	M_1	M_2	M_3	r_j
J_1	6	4	5	2
J_2	1	7	3	0
J_3	3	6	9	0
J_4	2	6	9	8

使用分支定界算法求解该问题的过程分别如图 2-3 和图 2-4 所示。在第 0 层，采用 SPTA 启发式求得根节点处的初始上界值为 $Z_0^{\mathrm{UB}}=2803$ (86 103)。在

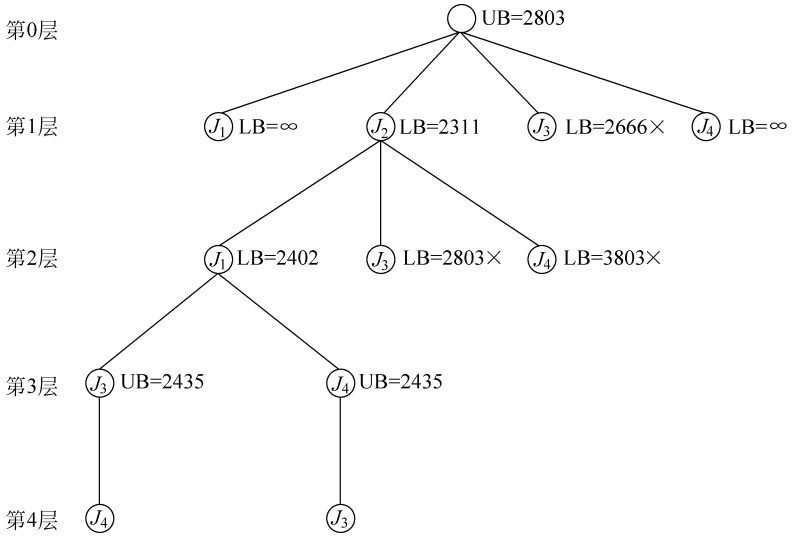

图 2-3　求解 $F_m|r_j|\sum C_j^2$ 问题的分支定界搜索树

×表示剪支

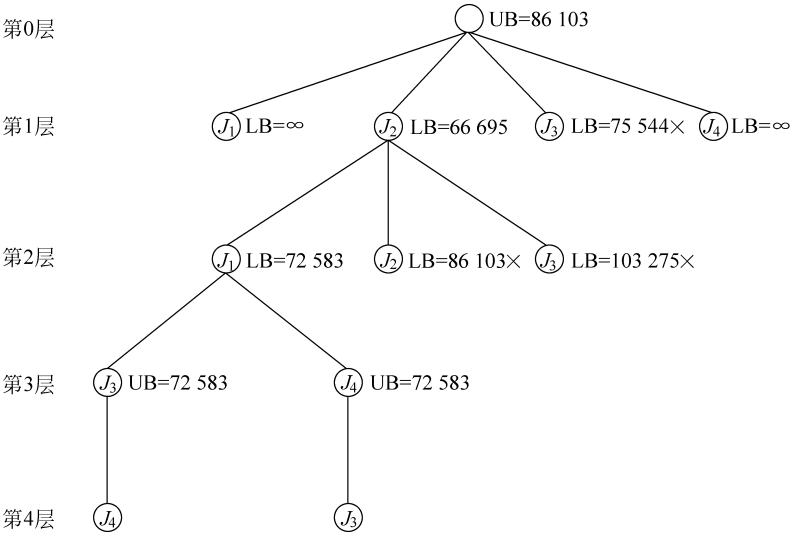

图 2-4　求解 $F_m|r_j|\sum C_j^3$ 问题的分支定界搜索树

×表示剪支

第 1 层，剪支规则直接剪掉节点 J_1 和 J_4。在第 3 层，更新 $Z^{UB}=2435(72\ 583)$。至此，所有剩余的有效节点都将被剪掉，最优调度为$\{J_2,J_1,J_3,J_4\}$或$\{J_2,J_1,J_4,J_3\}$。在此例中，只进行了 7 次计算，大大节省了求解时间。

2.4 非线性目标问题上界与下界的性能分析

2.3 节中,针对问题 2.3 设计了分支定界的算法下界 LB2.3,并采用 SPTA 启发式作为初始上界。本节将证明基于下界 LB2.3 设计的问题下界具有渐近最优性,并在一致性条件下对 SPTA 启发式的最坏性能进行了分析。

2.4.1 问题下界的收敛性分析

由于问题 2.3 具有 NP-难性质,在数值仿真中通常会采用问题下界代替最优解进行实验。为此,Bai 等(2014)提出了一种具有渐近最优性的下界 LB2.4:

$$Z_{2.4}^{\mathrm{LB}}=\max\{X_1,X_2,X_3\}$$

式中,$X_1=\sum_{j=1}^{n}\max_{1\leqslant x\leqslant j}\left\{r_x+\frac{1}{m}\sum_{i=1}^{m}\left(\sum_{v_1=1}^{i-1}p_{v_1,x}+\sum_{v=x}^{j}p_{i,v}+\sum_{v_2=i+1}^{m}p_{v_2,j}\right)\right\}^k$;

$X_2=\sum_{j=1}^{n}\max_{1\leqslant x\leqslant j}\left\{r_x+\sum_{h=x}^{j}p_{1,h}+\sum_{i=2}^{m}p_{i,j}\right\}^k$;

$X_3=\sum_{j=1}^{n}\max_{1\leqslant x\leqslant j}\left\{r_x+\sum_{i=1}^{m-1}p_{i,x}+\sum_{h=x}^{j}p_{m,h}\right\}^k$。

但是,下界 LB2.4 与任务序列相关,可能会出现下界值大于最优值的情况,具体见例 2-4。

例 2-4 考虑两台处理器的流水作业极小化完工时间平方和问题,其中包含两项任务$\{J_1,J_2\}$。相应的释放时间为 r_j,处理时间 $p_{i,j}(i=1,2,j=1,2)$ 如下所示:

	J_1	J_2	r_j
M_1	10	1	0
M_2	1	10	1

采用 SPTA 启发式得到任务调度为$\{J_1,J_2\}$,对应的下界值为

$$Z_{2.4}^{\mathrm{LB}}=(r_1+p_{1,1}+p_{2,1})^2+(r_1+p_{1,1}+p_{2,1}+p_{2,2})^2=562$$

显然,问题的最优调度为$\{J_2,J_1\}$,对应的最优值为

$$Z^{\mathrm{OPT}}=(r_2+p_{1,2}+p_{2,2})^2+(r_2+p_{1,2}+p_{1,1}+p_{2,1})^2=313$$

于是,有 $Z_{2.4}^{\mathrm{LB}}>Z^{\mathrm{OPT}}$。

为了保证问题下界的合理性,在分支定界算法下界 LB2.3 的基础上,本节构造了一种与任务序列无关的下界 LB2.5,其设计思想描述如下。将每台处理器上工序的调度看作是独立的 $1|r_j,\mathrm{prmp}|\sum C_j^k(k\geqslant 2)$ 问题,利用 SRPT 规

则求得最优解作为此台处理器的下界，然后在 m 个下界中选择目标值最大者作为原问题的下界。令 $r_{i,j}$ 表示任务 j 在处理器 i 上的可用时间，则 $r_{1,j}=r_j$，$r_{i+1,j}=r_{i,j}+p_{i,j}(i=1,2,\cdots,m-1;\ j=1,2,\cdots,n)$。于是，下界 LB2.5 的目标值表示为

$$Z_{2.5}^{\mathrm{LB}}=\max_{1\leqslant i\leqslant m}\left\{\sum_{j=1}^{n}(C_{i,j}^{\mathrm{SRPT}})^k+\sum_{x=1}^{k-1}\prod_{\alpha=1}^{x}\frac{k-(\alpha-1)}{\alpha}\left(\min_{1\leqslant j\leqslant n}\left\{\sum_{i'=i+1}^{m}p_{i',j}\right\}\right)^x\times\sum_{j=1}^{n}(C_{i,j}^{\mathrm{SRPT}})^{k-x}+\sum_{j=1}^{n}\left(\sum_{i'=i+1}^{m}p_{i',j}\right)^k\right\}$$

式中，若 $i+1>m$，则 $\sum_{i'=i+1}^{m}p_{i',j}=0$。

显而易见，下界 LB2.5 与任务序列无关，不会出现下界值大于上界值的情况。用其计算例 2-4 中的下界值，可得

$$Z_{2.5}^{\mathrm{LB}}=(C_{2,1}^{\mathrm{SRPT}})^2+(C_{2,2}^{\mathrm{SRPT}})^2=11^2+13^2=290<Z^{\mathrm{OPT}}$$

定理 2.1　设任务的处理时间 $p_{i,j}(i=1,2,\cdots,m;j=1,2,\cdots,n)$为独立可交换随机变量，释放时间 r_j 满足 $r_j=O(n)$，则有

$$\lim_{n\to\infty}\frac{Z_{2.5}^{\mathrm{LB}}}{n^{k+1}}=\lim_{n\to\infty}\frac{Z^{\mathrm{OPT}}}{n^{k+1}}(\mathrm{w.p.1})$$

这里 w.p.1 表示以概率 1(with probability 1)收敛。

证明　考虑 $1|r_j,\mathrm{prmp}|\sum C_j{}^k(k\geqslant 2)$问题，这里设任务的释放时间为 $r_{i,j}$，任务的处理时间为 $p_j=\frac{1}{m}\sum_{i=1}^{m}p_{i,j}\ (j=1,2,\cdots,n)$。令 $Z_1^{\mathrm{OPT}}=\sum_{j=1}^{n}(C_j^{\mathrm{SRPT}})^2$ 表示该问题的最优值，C_j^{SRPT} 表示任务 j 在 SRPT 规则下得到的完工时间值。显然，有

$$Z_1^{\mathrm{OPT}}<Z_{2.5}^{\mathrm{LB}}\tag{2-15}$$

在定理 2.1 的假设下，针对问题 2.3，Bai 等(2014)证明了

$$\lim_{n\to\infty}\frac{Z^{\mathrm{OPT}}-Z_1^{\mathrm{OPT}}}{n^{k+1}}=0(\mathrm{w.p.1})$$

结合不等式(2-15)，可得

$$0\leqslant Z^{\mathrm{OPT}}-Z_{2.5}^{\mathrm{LB}}<Z^{\mathrm{OPT}}-Z_1^{\mathrm{OPT}}\tag{2-16}$$

因此，将不等式(2-16)除以 n^{k+1}，并取极限可得

$$0\leqslant\lim_{n\to\infty}\frac{Z^{\mathrm{OPT}}-Z_{2.5}^{\mathrm{LB}}}{n^{k+1}}\leqslant\lim_{n\to\infty}\frac{Z^{\mathrm{OPT}}-Z_1^{\mathrm{OPT}}}{n^{k+1}}=0\tag{2-17}$$

重新整理式(2-17)即可得到定理结果。

2.4.2 初始上界最坏性能分析

针对问题2.3,分支定界算法的初始上界采用SPTA启发式获得,即每当首台处理器出现空闲,在可用任务中优先安排平均处理时间 $\left(p_j=\frac{1}{m}\sum_{i=1}^{m}p_{i,j}\right)$ 最小的任务。若此时无任务可用,则让首台处理器保持空闲,直到有任务释放为止。

Bai等(2014)证明了SPTA启发式对于问题2.3具有渐近最优性。然而,在某些极端情况下,该近似算法的求解效果很差。在带有释放时间的流水作业模型中,定义一致性条件(cons):即任务释放时间满足 $r_1\leqslant r_2\leqslant\cdots\leqslant r_n$,任务的处理时间满足 $p_j\leqslant p_{j+1}(1\leqslant j\leqslant n-1)$时。于是,能够得到如下结论。

定理 2.2 对于 $F_m|r_j,\text{cons}|\sum C_j{}^k(k\geqslant 1)$的任何实例,都有

$$\frac{Z^{\text{SPTA}}}{Z^{\text{OPT}}}\leqslant m^k \tag{2-18}$$

该比值为紧界。其中,Z^{SPTA} 表示SPTA启发式的目标值。

证明 考虑 $1|r_j,\text{prmp}|\sum C_j{}^k(k\geqslant 1)$问题,其中,任务的释放时间和加工时间分别为 r_j 和 $p_j=\frac{1}{m}\sum_{i=1}^{m}p_{i,j}$ $(j=1,2,\cdots,n)$。在一致性条件下,采用相邻任务交换方法,易证SPTA规则具有最优性。因此,单机可中断调度问题的SPTA规则可以作为 $F_m|r_j,\text{cons}|\sum C_j{}^k$ 问题的下界LB2.6。令 C_j^{LB} 表示下界中任务 j 的完工时间,则下界值可表示为

$$Z_{2.6}^{\text{LB}}=\sum_{j=1}^{n}(C_j^{\text{SPTA}})^k=\sum_{j=1}^{n}\left(\max_{1\leqslant x\leqslant j}\left\{r_x+\sum_{h=x}^{j}p_h\right\}\right)^k$$

且满足 $Z_{2.6}^{\text{LB}}<Z^{\text{OPT}}$。针对流水作业调度问题,令 C_j^{SPTA} 和 C_j^{OPT} 分别表示任务 j 在SPTA启发式与最优调度中的完工时间,则有

$$C_j^{\text{OPT}}\geqslant C_j^{\text{LB}}=\max_{1\leqslant x\leqslant j}\left\{r_x+\sum_{h=x}^{j}p_h\right\} \tag{2-19}$$

$$C_j^{\text{SPTA}}-r_x\leqslant m\cdot\max_{1\leqslant x\leqslant j}\left\{\sum_{h=x}^{j}p_h\right\} \tag{2-20}$$

结合不等式(2-19)和不等式(2-20),可得

$$C_j^{\text{SPTA}}\leqslant m\cdot\max_{1\leqslant x\leqslant j}\left\{r_x+\sum_{h=x}^{j}p_h\right\}\leqslant mC_j^{\text{OPT}} \tag{2-21}$$

对不等式(2-21)的两端取 k 次方,并对所有任务求和,可得

$$Z^{\text{SPTA}}\leqslant m^kZ^{\text{OPT}} \tag{2-22}$$

将不等式(2-22)重新整理即可得定理 2.2 的结论。

为了验证该比值为紧界，考虑一个 m 台处理器的流水作业调度问题，其中包括 mn 项任务：$J_1, J_2, \cdots, J_{mn}$。所有任务满足 $J_1 = J_2 = \cdots = J_n; J_{n+1} = J_{n+2} = \cdots = J_{2n}; \cdots; J_{(m-1)n+1} = J_{(m-1)n+2} = \cdots = J_{mn}$。任务的处理时间为 $p_{1,1} = p_{2,1} = \cdots = p_{n,1} = 1; p_{n+1,2} = p_{n+2,2} = \cdots = p_{2n,2} = 1; \cdots; p_{(m-1)n+1,m} = p_{(m-1)n+2,m} = \cdots = p_{mn,m} = 1$。这 mn 项任务的释放时间为 $r_1 = 0, r_2 = \varepsilon, \cdots, r_{mn} = (mn-1)\varepsilon$。其余任务的加工时间都为 ε，这里 ε 是(0,1)区间内任意小的正数。采用 SPTA 启发式得到的任务调度为 $\{J_1, J_2, \cdots, J_{mn}\}$，相应的目标值为

$$Z^{\mathrm{SPTA}} = \sum_{j=1}^{mn} [j + (m-1)\varepsilon]^2$$

最优调度为 $\{J_{(m-1)n+1}, J_{(m-1)n+1}, \cdots, J_1, J_{(m-1)n+2}, J_{(m-1)n+2}, \cdots, J_2, \cdots, J_n\}$，相应的最优值为

$$Z^{\mathrm{OPT}} = \sum_{j=1}^{n} \sum_{i=1}^{m} \{(m-1)n\varepsilon + j[(m-2+i)\varepsilon + 1]\}^2$$

由此可得

$$\begin{aligned}
&\frac{Z^{\mathrm{SPTA}}}{Z^{\mathrm{OPT}}} \\
&= \left[\frac{1}{6}(2m^3n^3 + 3m^2n^2 + mn) + (m^2 - m)(mn^2 + n)\varepsilon + m(m-1)^2 n\varepsilon^2\right] \div \\
&\quad \Big(\frac{1}{6}(2mn^3 + 3mn^2 + mn) + \frac{1}{2}(m^2 - m)(4n^3 + 5n^2 + n)\varepsilon + \\
&\quad \frac{1}{36}\{m(m-1)[(118m - 116)n^3 + (96m - 93)n^2 + (14m - 13)n]\}\varepsilon^2\Big) \to m^2
\end{aligned}$$

当 $n \to \infty, \varepsilon \to 0$。

2.5　混合离散差分进化算法

随着问题规模逐步扩大，分支定界算法的计算时间呈指数增加，很难在有限时间内找到问题的最优解。与其耗费大量运算时间寻找最优解，不如在限定时间内快速求得问题的近优解，这样既能提高复杂调度问题的求解效率，又能保证工业环境下生产的连续性。智能优化算法(metaheuristic algorithm)也称为元启发式算法，是一类高级启发式算法的总称，通过模拟自然或社会现象而得到，用以解决复杂优化问题。此类算法能够在计算资源受限、信息不完备情况下为优化问题提供足够好的可行解。不过，因为此类算法结构复杂，所以相

关理论研究结果较少。本章将分支定界算法与智能优化算法相结合，用于求解带有释放时间的流水作业调度模型。

差分进化(differential evolution，DE)算法是一种基于种群进化的随机搜索方法，用于求解复杂连续优化问题。在运行过程中，该算法对父代个体执行变异与交叉操作生成子代个体，通过竞争策略贪婪地选择其中的精英个体更新当前种群。DE 算法由 Storn 等(1997)提出，最初用于求解切比雪夫多项式，由于其具有结构简单、收敛速度快、鲁棒性强等优点，目前已受到越来越多的研究者关注。不过，经典的 DE 算法中个体采用浮点数编码方式，无法直接用于求解离散优化问题。Wang 等(2010)提出基于任务序列编码的离散差分进化(discrete DE，DDE)算法，用于求解同顺序流水作业调度问题。该编码方法非常简单：将染色体中的每个基因与可行调度中的每项任务建立一一对应关系。例如，染色体[5，2，3，6，4，1]代表给定可行调度中任务的处理顺序：$J_5 \rightarrow J_2 \rightarrow J_3 \rightarrow J_6 \rightarrow J_4 \rightarrow J_1$，这里 h 表示任务编号，$h=1,2,\cdots,6$，反之亦然。本节提出了一种混合离散差分进化(hybrid discrete differential evolution，HDDE)算法，下面介绍 HDDE 算法求解流水作业调度的详细过程。

1. 初始化

该阶段的任务是设置参数值并生成初始种群。种群规模记为 Λ，表示种群中个体的数目。在第 τ 次迭代过程中，第 h 个目标个体记为 $X_h^\tau=[x_{h,1}^\tau,x_{h,2}^\tau,\cdots,x_{h,n}^\tau]$，其中基因 $x_{h,j}^\tau$ 表示安排在第 j 个位置的任务；第 h 个变异个体记为 $V_h^\tau=[v_{h,1}^\tau,v_{h,2}^\tau,\cdots,v_{h,n}^\tau]$；第 h 个实验个体记为 $U_h^\tau=[u_{h,1}^\tau,u_{h,2}^\tau,\cdots,u_{h,n}^\tau]$；$h=1,2,\cdots,\Lambda$。当前最好个体记录最新种群中目标值最优的个体，当前最好个体随着种群的进化逐步更新，算法终止时的当前最好个体解码后即为最优解。初始种群中的一部分个体由分支定界算法生成，即在给定时间内运行分支定界算法，之后采用相应的启发式算法($C_{\max}$ 对应 LPTA 算法，$Q_{\max}$ 对应 LDTA 算法，$\sum C_j{}^k$ 对应 SPTA 算法)，将未剪支节点处的不完全序列补全为可行序列作为种群个体；另一部分个体由随机生成的可行序列构成，目的是保证初始种群的多样性，并且要避免两部分种群个体之间的重复。

2. 变异操作

种群中个体 X_h^τ 通过变异操作生成变异个体 V_h^τ，$h=1,2,\cdots,\Lambda$。在第 $\tau-1$ 代种群中，最好个体记为 $X_{\text{best}}^{\tau-1}=[x_{\delta,1}^\tau,x_{\delta,2}^\tau,\cdots,x_{\delta,n}^\tau]$。为了提高求解效率，这里采取用双差分个体扰动当前最好个体的方式进行变异：

$$V_h^\tau = X_{\text{best}}^{\tau-1} \oplus [Z \otimes (X_{\alpha_1}^{\tau-1} - X_{\beta_1}^{\tau-1}) \oplus Z \otimes (X_{\alpha_2}^{\tau-1} - X_{\beta_2}^{\tau-1})] \quad (2\text{-}23)$$

式中，$\alpha_x, \beta_x \in \{1,2,\cdots,\Lambda\}$是随机生成的整数；$x=1,2$；MP$\in(0,1)$为变异概率。令 $G_{h_x}^\tau = [g_{h_x,1}^\tau, g_{h_x,2}^\tau, \cdots, g_{h_x,n}^\tau]$表示差分个体，$x=1,2$。式(2-23)中，乘法⊗运算过程如下所示：

$$G_{h_x}^\tau = Z \otimes (X_{\alpha_x}^{\tau-1} - X_{\beta_x}^{\tau-1}) \Leftrightarrow g_{h_x,j}^\tau = (x_{\alpha_x,j}^{\tau-1} - x_{\beta_x,j}^{\tau-1})$$

$$= \begin{cases} x_{\alpha_x j}^{\tau-1} - x_{\beta_x,j}^{\tau-1}, & \text{rand}(\cdot) < Z \\ 0, & \text{其他} \end{cases}$$

式中，算法为每个差分个体基因 $g_{h_x,j}^\tau$ 生成一个随机数 rand(·)$\in(0,1)$。若此随机数小于变异概率，则保留差分个体分量，否则令该分量为 0。变异概率控制着差分个体的扰动程度，是差分进化算法的重要参数。式(2-23)中，加法⊕运算过程如下所示：

$$V_h^\tau = X_{\text{best}}^{\tau-1} \oplus (G_{h_1}^\tau \oplus G_{h_2}^\tau) \Leftrightarrow v_{h,j}^\tau = x_{\delta,j}^{\tau-1} \oplus (g_{h_1,j}^\tau \oplus g_{h_2,j}^\tau)$$

$$= \text{mod}[(x_{\delta,j}^{\tau-1} + (g_{h_1,j}^\tau + g_{h_2,j}^\tau) + n - 1), n] + 1 \quad (2\text{-}24)$$

式中，mod(·)表示取余数运算。式(2-24)中三个分量相加之后可能出现负值或超过 n，取余数的目的是将最终结果映射到$[1,n]$范围内。下面通过实例说明变异操作具体运算过程。

例 2-5　考虑一个 $n=5$ 的变异操作。表 2-1、表 2-3 是生成两个差分个体的过程。表 2-2、表 2-4 是乘法⊗运算过程，其中变异概率 MP=0.7，rand(·)$\in(0,1)$表示生成的随机数。表 2-5 是加法⊕运算过程，其中最后一行是取余运算后的结果。

表 2-1　变异操作生成第一个差分个体

个体	$(h,1)$	$(h,2)$	$(h,3)$	$(h,4)$	$(h,5)$
X_{α_1}	4	2	5	3	1
X_{β_1}	2	5	4	1	3
$X_{\alpha_1} - X_{\beta_1}$	2	−3	1	2	−2

表 2-2　第一个差分个体基因选取过程(MP=0.7)

个体	$(h,1)$	$(h,2)$	$(h,3)$	$(h,4)$	$(h,5)$
rand(·)	0.1	0.9	0.5	0.3	0.7
$X_{\alpha_1} - X_{\beta_1}$	2	−3	1	2	−2
G_{h_1}	2	0	1	2	0

表 2-3 变异操作生成第二个差分个体

个体	$(h,1)$	$(h,2)$	$(h,3)$	$(h,4)$	$(h,5)$
X_{α_2}	3	4	5	2	1
X_{β_2}	4	5	1	2	3
$X_{\alpha_2}-X_{\beta_2}$	−1	−1	4	0	−2

表 2-4 第二个差分个体基因选取过程(MP=0.7)

个体	$(h,1)$	$(h,2)$	$(h,3)$	$(h,4)$	$(h,5)$
rand(·)	0.5	0.2	0.6	0.3	0.8
$X_{\alpha_2}-X_{\beta_2}$	−1	−1	4	0	−2
G_{h_2}	−1	−1	4	0	0

表 2-5 变异操作中变异个体生成过程

个体	$(h,1)$	$(h,2)$	$(h,3)$	$(h,4)$	$(h,5)$
X_{best}	2	4	1	5	3
G_{h_1}	2	0	1	2	0
G_{h_2}	−1	−1	4	0	0
V_h	3	3	1	2	3

3. 交叉操作

该操作将变异个体 V_h^τ 与目标个体 $X_h^{\tau-1}$ 的基因片段混合之后产生新的实验个体 U_h^τ。首先,为变异个体中每个基因生成随机数 rand(·)∈(0,1),若此随机数小于交叉概率 CP∈(0,1),则保留对应的基因,否则删除。然后,采用双点插入(insert)策略,随机选择目标个体中的两个插入点,将选中的变异个体基因片段分为两部分插入目标个体中。最后,在所获得的交叉个体中删除重复的基因,即可得实验个体 U_h^τ,这里的实验个体对应一个可行解。下面接着例 2-5 说明交叉操作的具体过程。表 2-6 是在变异个体中选择基因片段的过程,其中基因 $v_{h,3}^\tau$ 被删除。图 2-5 显示了将变异个体基因片段插入目标个体的过程,其中生成的交叉个体中出现了基因重复现象。从左至右,删除重复的基因,得到实验个体。

表 2-6　交叉操作选择基因片段过程(CP=0.8)

个体	$(h,1)$	$(h,2)$	$(h,3)$	$(h,4)$	$(h,5)$
V_h	3	3	1	2	3
CP	0.4	0.1	0.8	0.7	0.5
V'_h	3	3	—	2	3

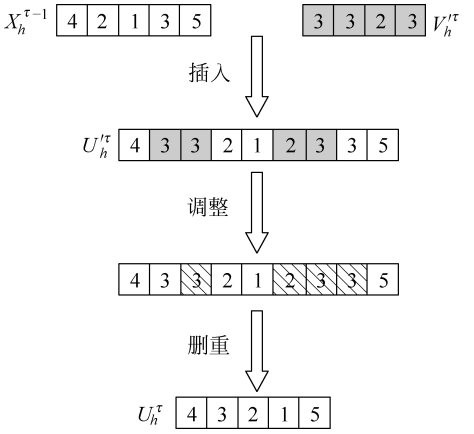

图 2-5　交叉操作中双点插入过程

4. 局部搜索

本节引入了局域搜索策略用以平衡算法的多样化和集约化从而提高解的质量。该操作通过破坏和重构来找到更好的解。在交叉个体的基础上执行局域搜索操作,为了节省计算时间,局域搜索操作以概率 θ 执行,以此增强对解的扰动并避免陷入局部最优。具体步骤描述如下所示。

步骤 1　破坏阶段。将候选解 π 分成两个队列:π_D 和 π_R,队列 π_D 由可行解 π 中随机选择的 φ 个任务(称为销毁长度)组成,队列 π_R 则由剩余任务组成。

步骤 2　重构阶段,π_D 中的每一项任务都被插入到队列 π_R 中的最佳位置。

步骤 3　选择阶段,分别计算重构后个体的目标函数与交叉个体的目标函数,若重构后的个体更优,则用重构个体替换当前交叉个体,否则保持交叉个体不变。

5. 选择操作

通过模拟大自然“物竞天择,适者生存”的规律,选择操作采用目标值更优的实验个体替换当前种群中的目标个体,保证下一代种群中的目标个体不会比

当前一代差。当整个种群中的个体都经历了变异、交叉和选择操作之后，就生成了新一代种群，同时将当前最好个体更新为当前种群中的最优个体。具体步骤描述如下所示。

步骤 1 令 $\tau=0$，$h=1$。将实验个体 U_h^τ 解码为可行调度 π_h^τ，并计算目标值 $Z(\pi_h^\tau)$。

步骤 2 若 $Z(\pi_h^\tau)<Z(\pi_h^{\tau-1})$，则令 $U_h^\tau=X_h^\tau$；否则，令 $X_h^{\tau-1}=X_h^\tau$，这里 $\pi_h^{\tau-1}$ 表示解码目标个体 $X_h^{\tau-1}$ 得到的可行调度。若 $h<\Lambda$，令 $h=h+1$，返回步骤 1；否则，转步骤 3。

步骤 3 若 $\tau<\tau_{\max}$，令 $\tau=\tau+1$，返回步骤 1；否则，终止迭代，这里 $\tau_{\max}$ 为最大迭代次数。

6. 算法流程

HDDE 算法的具体步骤描述如下所示。

步骤 1 产生初始种群。

(1)在给定时间内运行分支定界算法，在未剪支节点采用启发式调度未安排的任务生成可行调度。

(2)随机生成若干可行调度，与(1)中生成的可行调度，共同产生 Λ 个不相同的目标个体，作为初始种群，转到步骤 2。

步骤 2 设置算法参数. 确定最大迭代次数 $\tau_{\max}$、变异概率 MP 和交叉概率 CP。令迭代次数 $\tau=0$，在初始种群中找到当前最好个体 X_{best}。

步骤 3 令 $h=1$。

步骤 4 随机选择个体 X_α^τ 和 X_β^τ。

步骤 5 进行变异和交叉操作。

(1)产生变异个体 V_h^τ。

(2)采用基于双点插入的交叉操作，得到实验个体 U_h^τ，转到步骤 6。

步骤 6 对实验个体 U_h^τ 按照概率 θ 执行局部搜索操作，更新个体。

步骤 7 若 $h\leqslant\Lambda$，则令 $h:=h+1$，转到步骤 4；否则转到步骤 10。

步骤 8 执行选择操作，生成新一代种群，更新当前最好个体，转到步骤 9。

步骤 9 若 $\tau<\tau_{\max}$，则令 $\tau=\tau+1$，转到步骤 3；否则转到步骤 10。

步骤 10 终止程序，输出最优调度和最优值。

2.6 数值仿真实验

为了验证分支定界算法、智能优化算法以及问题下界的有效性，本节设计了不同规模的数值仿真实验，用于测试所提出算法与下界的求解性能。所有参

与测试的算法采用C++语言编写运行代码，并利用 Visual Studio 2010 进行编译。测试过程在安装 Intel® Core i7 CPU、8GB 内存、Windows® 7 (64 位)操作系统的电脑上进行。任务的处理时间 $p_{i,j}$ 由离散均匀分布[1,10]随机生成。释放时间 r_j 由离散均匀分布[0,5n]随机生成，不失一般性，保证其中至少有一项任务的释放时间为零。运送时间不能太短也不能太长，否则原问题就简化为了经典流水作业问题或车辆路径问题。因此，设定运送时间时必须确保 q_j 与 $p_{i,j}$ 之间不存在显著差异。运送时间 q_j 由离散均匀分布[$\min\{\eta,2n\},\max\{\eta,2n\}$]随机生成，其中 $\eta=\sum_{j=1}^{n}\sum_{i=1}^{m}p_{i,j}/(m\times n)$，$i=1,2,\cdots,n;j=1,2,\cdots,m$。测试结果及相关数据展示如下。

2.6.1　分支定界算法

本书成书时，还没有使用分支定界算法求解本章所讨论的三个流水作业问题的相关记录。为了显示分支定界算法求解实际问题的效果，这里将其与商业优化软件 IBM® CPLEX 12.8 进行对比实验，记录各自的运行时间与最终值。算法与求解器运行终止时间设定为 1h，若超过 1h，则终止运算记录当前最好目标值。

1. 最大完工时间问题

在测试中，处理器数为 $m=3,5,8$，任务数为 $n=10,12,15$。针对不同规模的处理器与任务组合，随机生成 5 组测试算例，采用本章所提出的分支定界算法与 CPLEX 求解器分别进行计算。从表 2-7 中展示的测试结果可以看出，分支定界算法在设定的求解时间内求得所有测试算例的最优解，其中约 71%(32/45)的算例在 0.1s 内完成最优求解；而 CPLEX 求解器仅求得其中约 53%(24/45)算例的最优解，其中约 33%(1/3)算例在(2,180)s 内求得最优解。可见，无论运行时间还是求解效率，分支定界算法都要显著优于 CPLEX 求解器。

2. 最大送达时间问题

最大送达时间问题测试算例规模与最大完工时间问题的实验完全相同。针对不同规模的处理器与任务组合，随机生成 5 组测试算例，采用本章所提出的分支定界算法与 CPLEX 求解器分别进行计算。从表 2-8 中展示的测试结果可以看出，在约 82%的算例中，分支定界算法效果明显优于 CPLEX 求解器。分支定界分别在 1s 和(1,60)s 内解决了约 47%(4/9)和 29%(13/45)的实例；而 CPLEX 求解器仅分别求得 7%(1/15)，42%(19/45)。显然，在求解效率上，分支定界算法优于 CPLEX 求解器。

表 2-7　最大完工时间问题精确求解对比实验

		$m=3$			$m=5$			$m=8$		
		B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX	B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX	B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX
$n=10$	Trail 1	79	79	0.016 ‖ 40.95	75	75	0.027 ‖ 48.72	110	110	0.023 ‖ 6.46
	Trail 2	72	72	0.006 ‖ 70.30	88	88	0.009 ‖ 83.04	133	133	0.031 ‖ 126.17
	Trail 3	90	90	0.009 ‖ 15.51	84	84	0.049 ‖ 2.20	105	105	0.016 ‖ 2.20
	Trail 4	84	84	0.007 ‖ 51.71	90	90	0.011 ‖ 2.22	107	107	0.027 ‖ 111.23
	Trail 5	85	85	0.010 ‖ 21.61	103	103	0.016 ‖ 29.22	96	96	0.051 ‖ 23.17
$n=12$	Trail 1	80	80	0.018 ‖ 2444.40	97	97	0.024 ‖ 3600	114	114	0.469 ‖ 3600
	Trail 2	88	88	0.009 ‖ 3600	94	94	0.037 ‖ 3600	111	111	0.175 ‖ 3600
	Trail 3	99	99	0.008 ‖ 3600	106	106	0.030 ‖ 1569.21	128	138	0.071 ‖ 2726.59
	Trail 4	88	88	0.061 ‖ 415.20	104	104	0.017 ‖ 299.43	127	127	0.016 ‖ 2838.63
	Trail 5	89	89	0.006 ‖ 313.06	114	114	0.041 ‖ 464.38	128	128	0.111 ‖ 1285.62
$n=15$	Trail 1	94	94	1.060 ‖ 3600	140	140	0.658 ‖ 3600	144	144	4.289 ‖ 3600
	Trail 2	115	115	0.011 ‖ 3600	115	115	0.067 ‖ 3600	133	136	2.199 ‖ 3600
	Trail 3	112	112	0.010 ‖ 3600	126	128	0.029 ‖ 3600	131	136	1.541 ‖ 3600
	Trail 4	109	110	0.017 ‖ 3600	118	118	0.220 ‖ 3600	166	169	0.362 ‖ 3600
	Trail 5	99	99	0.017 ‖ 3600	138	138	0.029 ‖ 3600	153	155	0.145 ‖ 3600

表 2-8　最大送达时间问题精确求解对比实验

		$m=3$			$m=5$			$m=8$		
		B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX	B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX	B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX
$n=10$	Trail 1	86	86	0.061 ‖ 3.29	113	113	0.117 ‖ 4.63	115	115	0.791 ‖ 4.68
	Trail 2	90	90	0.152 ‖ 20.3	95	95	0.029 ‖ 13.51	111	111	0.195 ‖ 1.45
	Trail 3	94	94	0.008 ‖ 0.98	97	97	0.073 ‖ 23.21	133	133	0.387 ‖ 44.23
	Trail 4	91	91	0.063 ‖ 3.20	123	123	0.059 ‖ 0.33	141	141	0.23 ‖ 5.27
	Trail 5	106	106	0.006 ‖ 0.48	100	100	0.231 ‖ 2.48	137	137	2.128 ‖ 8.24
$n=12$	Trail 1	109	109	0.444 ‖ 1271.50	118	118	2.312 ‖ 926.52	147	147	12.995 ‖ 419.27
	Trail 2	109	109	0.396 ‖ 1660.29	117	117	11.611 ‖ 99.70	140	140	10.623 ‖ 279.29
	Trail 3	102	102	1.1550 ‖ 143.93	119	119	1.039 ‖ 28.94	142	142	2.472 ‖ 35.40
	Trail 4	123	217	0.185 ‖ 3600	106	106	0.291 ‖ 34.74	152	152	2.460 ‖ 196.83
	Trail 5	126	126	0.511 ‖ 30.70	128	128	3.222 ‖ 1.83	159	159	28.146 ‖ 1015.38
$n=15$	Trail 1	135	130	3600 ‖ 3600	141	141	18.844 ‖ 757.57	165	168	3600 ‖ 3600
	Trail 2	115	115	2.407 ‖ 3600	141	140	3600 ‖ 1159.73	163	149	3600 ‖ 3600
	Trail 3	114	115	3600 ‖ 3600	143	140	3600 ‖ 3600	182	182	29.4 ‖ 3600
	Trail 4	154	154	28.869 ‖ 3600	163	163	0.640 ‖ 2.57	179	178	3600 ‖ 1160.57
	Trail 5	159	159	0.423 ‖ 33.46	164	146	3600 ‖ 3600	185	185	3600 ‖ 3600

3. 完工时间平方和问题

在被测试中，处理器数为 $m=3,5,8$，任务数为 $n=8,10,12,15$。针对不同规模的处理器与任务组合，随机生成 5 组测试算例，采用本章所提出的分支定界算法与 CPLEX 求解器分别进行计算。从表 2-9 中展示的测试结果可以看出，分支定界算法在设定的求解时间内求得约 95%算例的最优解，其中约 67%(2/3)的算例在 60s 内完成最优求解；而 CPLEX 求解器仅求得其中约 48%算例的最优解，其中约 32%(19/60)算例在 60s 内求得最优解。对于两者在 1h 内都未求得最优解的情况，分支定界算法求得的最终解明显优于 CPLEX 求解器。

分支定界算法之所以能够快速进行求解，主要得益于 2.3 节提出的剪支规则和分支定界算法下界有效地剪掉了大量无效分支节点。为了进一步说明剪支规则在算法求解过程中的重要作用，下面采用无剪支规则的分支定界(简称次分支定界)算法求解表 2-9 中的算例，最终计算结果记录在表 2-10 中。对比 CPU 时间可以看出，除了 3 个未能求得最优解的算例，次分支定界需要更多的运算时间。其中，大约有一半的算例，次分支定界需要多花费一半的 CPU 时间进行求解。特别地，若采用分支定界求解运行时间超过 100s 的算例，则将有 3/4 的算例可以节省 30%的求解时间。以上结果充分说明剪支规则在分支定界算法执行过程中删除了大量的无效节点，缩短了运算时间，提高了求解效率。

4. 完工时间立方和问题

完工时间立方和问题的测试算例规模与完工时间平方和问题完全相同。针对不同规模的处理器与任务组合，随机生成 5 组测试算例。因为 CPLEX 无法求解完工时间立方和目标函数，因此这里只列出分支定界算法的计算结果。从表 2-11 中展示的测试结果可以看出，分支定界算法在设定的求解时间内求得约 92%算例的最优解，而且，约 65%(39/60)的算例在 1min 内完成最优求解。表 2-11 与表 2-10 使用了相同的输入数据，对比平方和目标输出结果发现，在求解立方和目标时，算法花费的运算时间有所增加，说明在立方和实验中算法搜索了更多节点。引起该现象的原因可能是立方目标函数比平方目标扩大了下界与目标值之间的差异，导致某些平方和问题中可以被剪掉的节点在立方和问题中无法剪支，因此耗费了更多的搜索时间。

以上计算结果足以充分说明分支定界算法求解带有释放时间流水作业调度模型的高效性。但是，随着问题规模的进一步扩大，分支定界算法的求解时间呈指数增长，无法在规定时间内求得最优解。因此，对于中等规模问题，可以采用智能优化算法在短时间内获得高质量的可行解。

表 2-9　完工时间平方和问题精确求解对比实验

		$m=3$			$m=5$			$m=8$		
		B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX	B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX	B&B Objective	CPLEX Objective	CPU 时间/s B&B ‖ CPLEX
$n=8$	Trail 1	15 274	15 274	0.536 ‖ 2.36	17 451	17 451	0.881 ‖ 2.11	42 005	42 005	1.391 ‖ 0.98
	Trail 2	12 286	12 286	0.672 ‖ 2.67	25 823	25 823	1.113 ‖ 3073	46 286	46 286	0.740 ‖ 7.22
	Trail 3	9357	9357	0.136 ‖ 0.61	33 780	33 780	0.338 ‖ 0.67	48 112	48 112	0.495 ‖ 1.02
	Trail 4	14534	14 534	0.585 ‖ 3.09	24 691	24 691	1.239 ‖ 2.04	57 329	57 329	3.186 ‖ 5.43
	Trail 5	12 924	12 924	1.828 ‖ 4.88	34 166	34 166	0.732 ‖ 1.78	55 229	55 229	2.327 ‖ 4.7
$n=10$	Trail 1	29 350	29 350	3.216 ‖ 218.37	31 122	31 122	55.16 ‖ 899.31	76 155	76 155	15.81 ‖ 340.16
	Trail 2	22 546	22 546	1.567 ‖ 320.07	41 417	41 757	17.19 ‖ 3600	91 717	91 717	6.5 ‖ 586.95
	Trail 3	39 208	39 208	0.627 ‖ 39.26	33 403	33 403	3.26 ‖ 3.32	66 102	66 102	2.88 ‖ 39.27
	Trail 4	31 744	31 744	2.848 ‖ 242.52	41 494	41 494	1.58 ‖ 14.99	66 219	67 770	107.7 ‖ 3600
	Trail 5	28 111	28 111	0.274 ‖ 2.28	52 093	52 335	11.05 ‖ 260.54	52 670	52 670	38.57 ‖ 543.9
$n=12$	Trail 1	32 940	32 940	8.67 ‖ 3600	51 256	51 398	36.8 ‖ 3600	88 237	88 761	1282.56 ‖ 3600
	Trail 2	37 039	37 039	4.84 ‖ 3600	54 664	56 256	189.1 ‖ 3600	77 735	80 110	297.2 ‖ 3600
	Trail 3	48 035	48 035	10.14 ‖ 3600	62 575	62 757	7.87 ‖ 3600	115 172	144 157	351.9 ‖ 3600
	Trail 4	38 803	39 903	10.77 ‖ 3600	67 767	69 209	68.21 ‖ 3600	107 741	107 741	45.26 ‖ 3600
	Trail 5	36 977	36 977	1.3 ‖ 633.33	75 018	75 018	9.57 ‖ 3600	103 147	103 147	193.9 ‖ 3600
$n=15$	Trail 1	46 707	48 053	1467.2 ‖ 3600	141 320	141 320	314.8 ‖ 3600	161 465	169 929	3369 ‖ 3600
	Trail 2	87 934	88 277	3600 ‖ 3600	86 464	86 464	213.1 ‖ 3600	144 185	144 705	3600 ‖ 3600
	Trail 3	67 129	67 157	21.49 ‖ 3600	102 443	102 443	381.48 ‖ 3600	143 088	144 781	3600 ‖ 3600
	Trail 4	74 502	74 502	148.9 ‖ 3600	100 574	100 574	396.7 ‖ 3600	224 699	226 230	911.28 ‖ 3600
	Trail 5	70 456	72 050	32.46 ‖ 3600	126 798	127 892	72.4 ‖ 3600	181 950	181 950	548.27 ‖ 3600

表 2-10 无剪支规则分支定界算法测试实验(平方目标)

		$m=3$		$m=5$		$m=8$	
		Objectives	CPU 时间/s	Objectives	CPU 时间/s	Objectives	CPU 时间/s
$n=8$	Trial 1	15 274	0.802	17 451	1.674	42 005	2.696
	Trial 2	12 286	1.124	25 823	1.358	46 286	3.217
	Trial 3	9357	0.650	33 780	0.745	48 112	3.246
	Trial 4	14 534	0.689	24 691	2.270	57 329	4.673
	Trial 5	12 924	2.265	34 166	1.175	55 229	3.337
$n=10$	Trial 1	29 350	4.396	31 122	71.688	76 155	24.208
	Trial 2	22 546	3.080	41 417	20.628	91 717	9.789
	Trial 3	39 208	1.848	33 403	7.588	66 102	11.565
	Trial 4	31 744	3.660	41 494	3.399	66 219	139.958
	Trial 5	28 111	1.556	52 093	14.529	52 670	53.748
$n=12$	Trial 1	32 940	24.606	51 256	48.549	88 237	1557.291
	Trial 2	37 039	6.014	54 664	232.585	77 735	483.063
	Trial 3	48 035	13.147	62 575	12.999	115 172	464.100
	Trial 4	38 803	47.409	67 767	97.861	107 741	71.526
	Trial 5	36 977	2.819	75 018	17.540	103 147	272.000
$n=15$	Trial 1	46 707	2035.129	141 320	411.350	168 261	3600
	Trial 2	87 934	3600	86 464	311.269	144 185	3600
	Trial 3	67 129	34.597	102 443	512.981	143 088	3600
	Trial 4	74 502	185.183	100 574	900.645	224 699	2223.271
	Trial 5	70 456	51.562	126 798	101.914	181 950	927.595

表 2-11 完工时间立方和问题精确求解实验

		$m=3$		$m=5$		$m=8$	
		Objectives	CPU 时间/s	Objectives	CPU 时间/s	Objectives	CPU 时间/s
$n=8$	Trial 1	808 300	0.807	918 163	1.067	3 348 125	1.498
	Trial 2	610 007	1.129	1 554 369	1.162	3 990 548	0.954
	Trial 3	397 745	0.135	2 530 222	0.414	4 275 502	0.541
	Trial 4	746 279	0.806	1 500 129	1.541	5 096 051	3.490
	Trial 5	622 654	3.049	2 452 348	0.899	5 143 033	3.408

续表

		$m=3$		$m=5$		$m=8$	
		Objectives	CPU 时间/s	Objectives	CPU 时间/s	Objectives	CPU 时间/s
$n=10$	Trial 1	1 902 771	3.782	1 946 880	68.567	7 042 741	20.577
	Trial 2	1 286 511	2.407	3 049 743	25.695	9 911 137	9.780
	Trial 3	2 906 277	0.596	2 154 481	3.883	5 848 900	3.345
	Trial 4	2 166 452	3.454	3 040 260	2.006	6 097 207	173.692
	Trial 5	1 873 287	0.327	4 187 901	14.493	4 270 706	58.504
$n=12$	Trial 1	2 158 014	13.092	3 993 596	47.074	8 460 793	2032.049
	Trial 2	2 550 365	8.332	4 201 458	388.550	7 224 929	488.255
	Trial 3	3 694 245	14.479	5 244 244	10.608	12 320 490	457.038
	Trial 4	2 747 621	19.302	5 708 069	102.018	11 213 219	56.251
	Trial 5	2 531 441	1.606	6 899 139	15.879	10 674 535	338.851
$n=15$	Trial 1	3 430 153	3600	15 517 909	477.387	20 365 284	3600
	Trial 2	8 144 046	3600	7 553 403	236.775	16 081 297	3600
	Trial 3	5 555 581	28.924	10 084 855	601.271	15 865 924	3600
	Trial 4	6 169 046	34.587	9 431 208	462.567	31 130 633	1277.435
	Trial 5	5 665 535	72.975	13 451 490	84.596	22 965 126	1132.545

2.6.2　离散差分进化算法

智能优化算法的寻优性能很大程度上取决于参数的设置，因此，在求解之前需要通过正交实验确定合适的参数。处理器与任务的测试规模分别设置为 $m=3,5,10$ 与 $n=60,100,140,180$。对于设定的每种测试规模（$m\times n$），分别生成 10 组测试数据（包括处理时间、释放时间、运送时间）。针对每组测试数据，智能优化算法分别进行 5 次随机实验。为了验证算法的有效性，分别采用了平均相对误差（mean relative gap，MRG）与相对差异百分比（releative difference percentage，RDP）作为评价指标。其中，MRG 的表达式为

$$\mathrm{MRG}=\frac{Z^{\mathrm{INI}}-Z^{\mathrm{FIN}}}{Z^{\mathrm{FIN}}}\times 100\%$$

式中，Z^{INI} 与 Z^{FIN} 分别表示算法的初始值与最终值。

RDP 的表达式为

$$\mathrm{RDP}=\frac{Z^{H}-Z^{*}}{Z^{*}}\times 100\%$$

式中，Z^H 表示算法求得的最终值；Z^* 表示所有 Z^H 值中的最好值。

1. 最大完工时间问题

根据正交实验的结果，HDDE 算法求解问题 2.1 的参数设置为：种群规模 $\Lambda=100$，变异概率 MP=0.2，交叉概率 CP=0.4，局域搜索概率 $\theta=0.4$，扰动步长 $\varphi=5$，最大迭代次数 $\tau_{\max}=150$。释放时间由 r_j 分别由离散均匀分布[0，50n]随机生成。处理时间是取自流水作业基准集数据(Taillard，1993)。

表 2-12 中的数据是 HDDE 算法求解问题 2.1 的 MRG 值。实验结果表明，对于给定的处理器规模，随着任务数量的增加，HDDE 算法的优化性能有所下降。例如，5 台处理器时，当任务数由 60 增加到 180 时，MRG 值由 32.5637%增长到 38.8509%。引起该现象的原因可能是问题规模增大引起分支定界生成的初始种群质量下降，导致初始解与最终解之间的误差变大。

表 2-12 HDDE 算法求解问题 2.1 的 MRG 值 %

	$m=3$	$m=5$	$m=10$
$n=60$	33.2797	32.5637	33.3717
$n=100$	36.1156	36.4508	35.9964
$n=140$	37.0911	37.6736	37.748
$n=180$	38.2047	38.8509	38.9296

为了突出 HDDE 算法的优良性能，表 2-13 记录了 HDDE 与 GA(见附录 A.1)求解问题 2.1 的对比实验结果。测试中 $m=5$，10，20 和 $n=20$，50，100，200。其余参数与 2.6.1 节中生成方式相同。表 2-13 中，MRDP、MinRDP、MaxRDP、和 SD 分别表示平均 RDP、最小 RDP、最大 RDP 和标准差。从表 2-13 可知，HDDE 算法的 MRDP 值明显优于 GA。例如，遗传算法的 MRDP 值的均值为 47.781%，而 HDDE 算法的相应值为 3.414%。而且，前者的 SD 值明显大于后者，这体现了 HDDE 算法的稳定性。

表 2-13 HDDE 与 GA 求解问题 2.1 结果比较

$n\times m$	GA				HDDE			
	MRPD	MinRPD	MaxRPD	SD	MRPD	MinRPD	MaxRPD	SD
20×5	36.851	9.934	69.643	16.029	8.634	0	33.333	11.941
20×10	31.881	8.602	52.632	14.982	7.725	0	30.921	10.2
20×20	16.732	6.792	36.726	7.692	1.291	0	15.929	3.227
50×5	41.674	33.724	53.443	5.169	0.339	0	1.27	0.375

续表

$n\times m$	GA				HDDE			
	MRPD	MinRPD	MaxRPD	SD	MRPD	MinRPD	MaxRPD	SD
50×10	40.858	33.423	48.889	4.482	0.323	0	1.596	0.45
50×20	35.530	28.785	41.475	3.167	0.282	0	0.922	0.318
100×5	52.856	47.687	61.512	3.629	0.308	0	1.313	0.394
100×10	52.794	45.269	58.886	3.288	0.317	0	2.108	0.494
100×20	46.179	41.722	51.579	2.338	0.342	0	1.028	0.357
200×10	85.716	68.812	96.601	7.517	10.182	0	20.963	8.3
200×20	84.517	57.184	101.636	14.145	7.814	0	24.801	9.932
平均值	47.781	34.721	61.184	7.494	3.414	0	12.182	4.181

2. 最大送达时间问题

根据正交实验的结果，HDDE 算法求解问题 2.2 的参数设置为：种群规模 $\Lambda=100$，变异概率 MP=0.2，交叉概率 CP=0.2，局域搜索概率 $\theta=0.4$，扰动步长 $\varphi=5$，最大迭代次数 $\tau_{\max}=150$。实验数据与最大完工时间问题一致，运送时间由均匀分布 $U[\bar{p}m,\bar{p}m+10n]$生成，其中 $\bar{p}=\sum_{j=1}^{n}\sum_{i=1}^{m}p_{i,j}/(mn)$。

表 2-14 中的数据是 HDDE 算法求解问题 2.2 的 MRG 值。实验结果显示了与前一节相同的变化趋势，这可能是由相同的原因引起的。同时，该表中的数据值略大于表 2-12 中的值，这说明 HDDE 算法求解问题 2.2 的效率更高。

表 2-14　HDDE 求解问题 2.2 的 MRG 值　%

	$m=3$	$m=5$	$m=10$
$n=60$	34.9610	36.1909	35.5257
$n=100$	37.6436	38.4527	39.0593
$n=140$	38.0885	39.0291	40.2686
$n=180$	39.4923	39.9370	40.7665

为了突出 HDDE 算法的优良性能，表 2-15 记录了 HDDE 与 GA 算法求解问题 2.2 的对比实验结果。参数与 2.6.1 节中生成方式相同。表 2-15 中实验结果显示的总体趋势与表 2-13 中数据基本一致，这验证了 HDDE 算法在运算性能上要优于 GA。

表 2-15 HDDE 与 GA 求解问题 2.2 结果比较

$n\times m$	GA				HDDE			
	MRPD	MinRPD	MaxRPD	SD	MRPD	MinRPD	MaxRPD	SD
20×5	19.444	4.372	52.000	11.622	2.326	0	36.800	7.099
20×10	19.170	9.278	34.054	6.311	2.188	0	12.169	4.200
20×20	12.608	6.071	19.718	3.421	0.323	0	1.418	0.435
50×5	46.748	38.483	52.616	3.531	0.103	0	0.560	0.156
50×10	47.256	37.221	61.247	5.486	0.878	0	10.569	2.541
50×20	41.406	33.673	48.000	3.534	0.263	0	0.844	0.294
100×5	58.655	46.676	67.862	5.187	0.439	0	4.160	1.041
100×10	59.804	52.597	66.289	4.022	0.321	0	0.992	0.342
100×20	56.301	51.381	60.979	2.163	0.229	0	0.963	0.269
200×10	71.441	61.783	84.161	7.143	1.674	0	11.669	3.495
200×20	76.199	62.647	97.280	9.512	2.903	0	16.084	4.842
平均值	46.276	36.744	58.564	5.630	1.059	0	8.748	2.247

3. 完工时间 k 次方和问题

DDE 算法求解问题 2.3 的参数设置为：种群规模 $\Lambda=n$，变异概率 MP=0.2，交叉概率 CP=0.2，最大迭代次数 $\tau_{\max}=300$。任务的处理时间除了与前一节相同的离散均匀分布之外，还随机取自于期望为 5.5 标准差为 1.7 的正态分布。初始种群采用分支定界算法生成，交叉操作中采用三点插入策略。

表 2-16 和表 2-17 中的数据表明，DDE 算法的优化性能与处理时间的分布无关。当任务数固定时，算法的求解效率随着处理器数量的增加而逐渐增强。对于 100 项任务、处理时间为正态分布、目标函数为平方(立方)和的实例，当处理器数目由 3 台增加到 10 台时，MRG 值由 7.31%(9.95%)变为 11.65%(14.98%)。这种现象可能是处理器数目增加引起每项任务的工序数目增加，导致空闲时间增多从而降低了初始解的质量。当处理器数固定时，算法的求解效率随着任务数目的增加而逐渐减弱。此外，对于 5 台处理器、处理时间为均匀分布、目标函数为平方(立方)和的实例，当任务数由 60 项增加到 180 项时，MRG 值由 20.54%(19.92%)变为 9.06%(12.02%)。这种现象可能是可用任务数目增加使得每项任务被处理器加工的机会增加，这样就减少了处理器的等待时间从而增强了初始解的质量。

表 2-16　DDE 算法求解平方和目标问题　%

	均匀分布			正态分布		
	$m=3$	$m=5$	$m=10$	$m=3$	$m=5$	$m=10$
$n=60$	13.62	20.54	20.25	8.59	11.15	15.37
$n=100$	7.39	13.11	14.72	7.31	11.43	11.65
$n=140$	8.94	11.39	12.36	6.48	9.55	12.07
$n=180$	6.98	9.06	12.52	5.15	9.14	10.41

表 2-17　DDE 算法求解立方和目标问题　%

	均匀分布			正态分布		
	$m=3$	$m=5$	$m=10$	$m=3$	$m=5$	$m=10$
$n=60$	23.49	19.92	26.09	13.08	16.61	19.72
$n=100$	10.51	19.26	21.08	9.95	12.74	14.98
$n=140$	10.48	13.37	18.68	5.77	10.81	12.73
$n=180$	5.83	12.02	13.25	6.99	8.74	13.52

为了突出 DDE 算法的优良性能，将其与 PSO 算法（见附录 A.2）进行了对比实验。其中，释放时间 r_j 由离散均匀分布[0,3n]随机生成，并保证其中至少有一项任务的释放时间为零。任务的处理时间除了与前一节相同的离散均匀分布之外，还随机取自于期望为 15 标准差为 5 的正态分布。根据正交实验的结果，DDE 算法求解问题 2.3 的参数设置为：种群规模 $\Lambda=n$，变异概率 MP=0.2，交叉概率 CP=0.2，最大迭代次数 $\tau_{\max}=300$。PSO 算法求解问题 2.3 的参数设置为：惯性权重=0.7，最大速度=1，最大位置=4，认知系数=2.5，社会系数=1.0。实验结果采用 GAP 值表示：$\mathrm{GAP}=\dfrac{Z^{\mathrm{PSO}}-Z^{\mathrm{DDE}}}{Z^{\mathrm{DDE}}}\times 100\%$，这里 Z^{PSO} 与 Z^{DDE} 分别表示 PSO 与 DDE 算法的最终解的目标值。

表 2-18 和表 2-19 记录了 DDE 与 PSO 算法求解问题 2.3 的对比实验数据。两个表中 GAP>0，说明 DDE 算法的最终目标值优于 PSO 算法。而且，对于相同的处理器数量，随着问题规模增大，GAP 值也在逐渐增加，说明在求解过程中，DDE 算法的稳定性比 PSO 算法更强。

表 2-18　平方和目标问题 DDE 与 PSO 算法对比实验结果

	均匀分布			正态分布		
	$m=3$	$m=5$	$m=10$	$m=3$	$m=5$	$m=10$
$n=60$	2.0598	3.708	1.2937	0.9791	0.4149	1.8643

续表

	均匀分布			正态分布		
	$m=3$	$m=5$	$m=10$	$m=3$	$m=5$	$m=10$
$n=100$	4.616	4.6731	4.0158	3.6059	0.2318	1.7175
$n=140$	8.1077	7.2033	5.1449	3.796	0.7024	1.1464
$n=180$	9.9647	10.224	5.7773	4.5267	3.1844	2.7471

表 2-19 立方和目标问题 DDE 与 PSO 算法对比实验结果

	均匀分布			正态分布		
	$m=3$	$m=5$	$m=10$	$m=3$	$m=5$	$m=10$
$n=60$	2.6956	4.0658	1.1522	0.0023	0.9731	0.0012
$n=100$	7.0116	5.3591	1.8383	0.478	0.0363	0.1183
$n=140$	8.6981	11.763	33.918	1.646	1.9433	0.914
$n=180$	12.416	31.974	6.9624	1.7686	0.0351	0.7523

2.6.3 问题下界性能测试

为了检验下界 LB2.5 处理问题 2.3 时的收敛性，本节设计了如下数值实验。任务与处理器的测试规模分别设置为 $m=3,5,10$ 与 $n=100,500,1000$。对于设定的每种测试规模($m\times n$)，分别生成 10 组测试数据(包括处理时间、释放时间)，并记录最终结果的平均值。任务的释放时间设置为：$r_1=0$；$r_{j+1}=r_j+y$；$j=2,3,\cdots,n$；这里 y 取值于离散均匀分布[1,10]的随机变量。由于问题 2.3 具有 NP-难性质，即使求解小规模问题也只能应用基于枚举的精确算法。Bai 等(2014)证明了 SPTA 启发式具有渐近最优性，因此，这里将其作为问题 2.3 最优解的替代值。平均相对误差(mean relative error，MRE) $\alpha_1=\dfrac{Z^{\mathrm{ALG}}-Z^{\mathrm{LB}}}{Z^{\mathrm{LB}}}$ (其中，Z^{ALG} 表示 SPTA 启发式得到的目标值；Z^{LB} 表示下界值)与平均相对比(mean relative ratio，MRR) $\alpha_2=\dfrac{Z^{\mathrm{LB2.4}}}{Z^{\mathrm{LB2.5}}}$ 分别用来衡量下界的收敛性和优越性。

表 2-20 和表 2-21 中的实验结果显示，对于固定的处理器数，随着任务规模的增加 α_1 值逐渐减小，这说明下界 LB2.5 收敛到 SPTA 启发式，从侧面验证了下界的渐近最优性。对于 5 台处理器、目标函数为平方(立方)和的实例，当任务数由 100 项增加到 1000 项时，α_1 值由 0.2921(0.2977)减小到 0.0774(0.1110)，这充分证实了定理 2.1 的正确性。此外，对于固定的任务数，随着处理器(幂指数)规

模的增加 α_1 值逐渐增大，这说明下界 LB2.5 的收敛性与处理器（幂指数）规模相关。对于 500 项任务、目标函数为平方（立方）和的实例，当处理数由 3 台增加到 10 台时，α_1 值由 0.0564(0.1091)增加到 0.2318(0.3051)。这种现象可能是任意相邻两项任务之间的空闲时间会随着处理器（幂指数）规模扩大而增加，这导致了启发式与下界之间的误差变大。两张表中的 α_2 值显示在大多数情况下，下界 LB2.5 优于下界 LB2.4。在立方和目标问题中，当任务数为 1000，处理器分别为 5 台和 10 台时，α_2 值小于 1，这说明下界 LB2.4 在处理大规模问题时稳定性较差。当任务数充分大时，两个下界的目标值趋于相同。

表 2-20　平方目标问题下界 2.5 实验结果

	$m=3$			$m=5$			$m=10$		
n	100	500	1000	100	500	1000	100	500	1000
α_1	0.1612	0.0564	0.0479	0.2921	0.1231	0.0774	0.3933	0.2318	0.1570
α_2	1.0274	1.0082	1.0104	1.0493	1.0032	1.0077	1.0538	1.0154	1.0057

表 2-21　立方目标问题下界 2.5 实验结果

	$m=3$			$m=5$			$m=10$		
n	100	500	1000	100	500	1000	100	500	1000
α_1	0.1615	0.1091	0.0645	0.2977	0.1901	0.1110	0.6014	0.3051	0.2409
α_2	1.0521	1.0102	1.0119	1.0376	1.0088	0.9959	1.0465	1.0229	0.9989

2.6.4　工业数据测试

变速箱是风力发电机的重要机械部件，其主要作用是将风轮在风力吹动下产生的能量传递给发电机，从而获得相应的转速。根据真实数据的统计分析的结果，在装配作业变速箱的生产过程主要由 8 个串联步骤构成：

（1）装配件清洗——30min；

（2）齿轮与轴承热装——U[660,900]min；

（3）热装后产品冷却——30min；

（4）冷却后产品调试——U[540,660]min；

（5）产品功能检测——U[60,120]min；

（6）润滑剂注入——U[18,30]min；

（7）产品外壳喷漆——U[7,14]min；

（8）附属零件组装——U[18,38]min。

这里 U[·,·]表示均匀分布。显然，变速箱的生产过程是典型的 8 机流

水作业调度模型，可以将其模拟为一个工业生产环境，用来评价非线性目标的效果以及 DDE 算法的性能。

1. 非线性目标函数优化效果

对于该调度模型的相同实例，分别讨论目标函数 $C_{\max}$ 和 $\sum C_j^{\,k}$，其中 $k=1,2,3$。对于 $C_{\max}$ 和 $\sum C_j$ 目标，利用 CPLEX 求解器分别求得最优调度 π_1 和 π_2。相应地，对于 $\sum C_j^{\,2}$ 和 $\sum C_j^{\,3}$ 目标，用分支定界算法分别获得最优调度 π_3 和 π_4。对于每种目标函数，分别测试了 8、10、12 项任务。释放时间随机生成于 $[0,700n]$ 离散均匀分布，其中要求至少有一项任务的释放时间为零。平均相对百分比 $\alpha_3=\dfrac{C_{\max}(S_\lambda)-C_{\max}(S_1)}{C_{\max}(S_1)}\times 100\%$ 和 $\alpha_4=\dfrac{\sum C_j(S_\lambda)-\sum C_j(S_2)}{\sum C_j(S_2)}\times 100\%$ 作为目标函数测试指标，其中 $\lambda=3,4$。每种问题规模进行 5 组随机测试，并记录最终结果。

表 2-22 中有限的数据表明，非线性目标函数（即 $\sum C_j^{\,2}$ 和 $\sum C_j^{\,3}$）对最大完工时间和完工时间和的影响几乎相同。对于大约 87%的实例，非线性目标求得的最优解与完工时间和目标完全相同。然而，大部分的非零 α_3 值说明非线性目标和最大完工时间目标之间存在一些差异。这可能是因为一个调度实例可能存在多个最优解，而计算最大完工时间目标值时只选择了其中的一个最优解。但是，这个误差微不足道（<1%），可以忽略不计。因此，可以认为利用非线性目标得到的最优调度可以同时优化最大完工时间与完工时间和两个目标函数。

表 2-22 非线性目标函数优化效果测试 %

		平方和		立方和	
		α_3	α_4	α_3	α_4
$n=8$	Trial 1	0	0	0	0
	Trial 2	0	0	0	0
	Trial 3	0	0	0	0
	Trial 4	0.0140	0	0.0140	0
	Trial 5	0	0	0	0
	平均值	0.0028	0	0.0028	0

续表

		平方和		立方和	
		α_3	α_4	α_3	α_4
$n=10$	Trial 1	0	0	0	0
	Trial 2	0.6352	0	0.6352	8.1937
	Trial 3	0	0	0	0
	Trial 4	1.6689	0	1.6689	0
	Trial 5	0	0	0	0
	平均值	0.4608	0	0.4608	1.6387
$n=12$	Trial 1	0.8668	0	0.8668	0
	Trial 2	0.1143	0	0.1143	0.0766
	Trial 3	0.2022	0	0.2022	0
	Trial 4	0.8733	0	0.8733	0
	Trial 5	1.2023	0	1.2023	0
	平均值	0.6518	0	0.6518	0.0153

2. DDE 算法性能测试

对于相同的初始种群，将 DDE 与 PSO 算法进行对比，突出前者的优化性能。PSO 算法的关键步骤可参见 Ren 等(2017)的文章，其中参数设置为：惯性系数 $w=0.3$，认知系数 $c_1=2.0$，社会系数 $c_2=2.0$，粒子大小 $\Lambda'=5n$，最大迭代 $\tau_{\max}=300$；每个粒子的位置和速度分别在[0,1]和[−4,4]上初始化。释放时间随机生成于离散均匀分布[0,500n]，其中要求至少有一项任务的释放时间为零。每种问题规模进行 10 次随机测试，并记录最终的平均值。

表 2-23 中的 GAP 值显示，平方和目标与立方和目标的差值范围分别为(47.5±35.6)%和(38.1±25.7)%。这些结果表明，DDE 算法的性能明显优于 PSO 算法。随着任务数量的增加，PSO 算法的优化性能下降幅度大于 DDE 算法。该现象可能是因为前者的进化机制较差，导致迭代过程中出现早熟。

表 2-23　DDE 算法工业数据测试　%

n	50	100	150	200
平方和	11.930	35.025	58.953	83.091
立方和	12.413	34.969	49.396	63.609

2.7 本章小结

本章研究了考虑释放时间的流水作业调度模型，其中目标函数分别为极小化最大完工时间、最大送达时间与完工时间 k（$k\geqslant 2$）次方和。针对完工时间 k 次方和的大规模问题，在概率极限意义下和一致性条件下提出了基于 SPTA 的启发式算法，并分析了该算法的渐近性能与最坏性能。数值仿真结果表明，当问题规模足够大时，该启发式可以用作最优算法，不仅能够快速求解，而且可以保证调度过程的连续性。针对小规模问题，设计了 B&B 算法进行精确求解，其中基于可中断的下界以及基于不同目标优势规则的剪支策略显著缩小了搜索空间。数值实验证实 B&B 算法的性能明显优于 CPLEX 求解器。针对中等规模问题，采用混合离散差分进化算法求得高质量可行解。对比实验和工业数据测试均表明，所提出的算法显著优于流行的遗传算法与粒子群优化算法。此外，得益于优化算法较好的泛化性，本节所提出的算法可以进一步推广，用于求解工业生产中具有更复杂约束的调度问题。

第 3 章　考虑处理器阻塞的流水作业调度问题

3.1　引言

流水作业调度是目前研究最为广泛的组合优化模型之一，被广泛应用于钢铁生产、汽车制造等行业。为了理论研究方便，通常假设相邻两台处理器之间的缓存容量为无穷大。然而，在实际调度环境中，受在制品体积的影响，缓冲区容量是有限的。一旦在制品充满缓冲区，在上游处理器已完工的任务就无法进入缓冲区等待，只能继续占用该处理器直至下游处理器空闲为止，称此现象为阻塞。在实际生产中，任务体积较大或工序之间需要运输衔接的情况下会发生阻塞现象。例如：大型闸阀是油田、化工厂主要生产设备的关键部件，其制造过程主要包括铸坯、热处理、焊接部件、无损检测和总装配五道工序；生产设备按照相同的工艺路线加工阀门半成品，由于阀门体积较大，因此，存储在制品的缓冲区受限。大型闸阀的制造过程可归为典型的阻塞流水作业调度(blocking flowshop scheduling，BFS)模型，即多项不同的任务按照相同的工艺路线经过多台串联处理器加工，其中缓冲区容量有限，目标为最优化给定的调度准则。

为了行文简洁，以下采用标准的三参数表示法来描述调度问题。Lenstra 等 (1977)指出 $F_2|r_j|C_{\max}$ 问题为强 NP-难。因为两台机器的问题是 $F_m|r_j$，block$|C_{\max}$、$F_m|r_j$，block$|L_{\max}$ 及 $F_m|r_j$，block$|Q_{\max}$(分别称为问题 3.1、问题 3.2 和问题 3.3)问题的特殊情况，所以后三个问题至少和前一个问题一样复杂，同样为强 NP-难。对于 NP-难问题，通常采用精确算法寻找最优解，目的是建立一个基准测试集来评价近似算法的性能。表 3-1 中列出了过去 50 年中采用分支定界算法求解 BFS 问题的一些结果。

表 3-1　求解 BFS 问题的分支定界算法

问题	参考文献
$F_m\|\text{block}\|C_{\max}$	Levner (1969)，Suhami 等(1981)，Ronconi (2005)，Companys 等(2007)，Toumi 等(2017)
$F_m\|\text{block},\alpha p_{i,j}\|C_{\max}$	Lee 等(2010)
$F_m\|\text{block}\|\sum C_j$	Moslehi 等(2013)
$F_m\|\text{block}\|\sum T_j$	Ronconi 等(2001)

然而，这些分支定界算法是依赖于特定条件的，不能推广到带有释放时间的 BFS 模型中。虽然分支定界算法可以在一定的时间内求解小规模实例，但是本质上是基于枚举的算法，随着问题规模扩大，其运行时间呈指数增长。相比之下，在较短时间内采用智能优化算法获得中等规模实例的近似解则是一种非常实用的求解方法。表 3-2 中列出了采用基于单点搜索和种群进化的智能优化算法求解 BFS 问题的一些结果。然而，这些智能优化算法大多只能用于求解任务同时可用的 BFS 模型。若采用智能优化算法解决大规模实例，其迭代运算也需要较多的计算时间，因此本章考虑使用调度规则求得大规模问题的可行解（详见表 3-3）。由于调度规则结构简单，可以对其进行理论性能分析（即最坏情况分析和渐近性能分析）。

表 3-2 求解 BFS 问题的智能优化算法

问题	算法	参考文献
$F_m\|\text{block}\|C_{\max}$	GA	Caraffa 等(2001)
	TS	Grabowski 等(2007)
	SA	Companys 等(2010)，Wang 等(2012)
	HS	Wang 等(2011)
	DE	Wang 等(2010)，Davendra 等(2012)
	PSO	Liang 等(2011)，Wang 等(2012)
	IG	Ribas 等(2011)，Ding 等(2015)，Tasgetiren 等(2017)
	ABC	Han 等(2012)，Han 等(2015)
	SMOA	Davendra 等(2013)
	ISA	Lin 等(2013)
	MMA	Pan 等(2013)
	VNS	Ribas 等(2013a)
	IBBO	Liu 等(2018)
	FFOA	Han 等(2016)
$F_m\|\text{block},t_j\|C_{\max}$	GA	Carlier 等(2010)
$F_m\|\text{block}\|\sum C_j$	HS	Wang 等(2010)
	ABC	Deng 等(2012)，Han 等(2013)，Ribas 等(2015)
	IG	Khorasanian 等(2012)
	GRASP	Ribas 等(2015)
$F_m\|\text{block}\|\sum T_j$	GRASP	Ronconi 等(2009)
	ILS	Ribas 等(2013b)
	CSA	Nagano 等(2019)
	IWO	Shao 等(2017)

表 3-3　求解 BFS 问题的调度规则

问题	参考文献
$F_m \mid \text{block}, \text{stant} \mid C_{\max}$	Reddi 等(1972) Logendran 等(1993)
$F_m \mid \text{prmu}, \text{no-wait}, \text{block}, b_{j,j+1} \mid C_{\max}$	Liu 等(2009)
$F_2 \mid \text{no-wait}, \text{block}, \text{batch}, \text{sharedst} \mid C_{\max}$	Gong 等(2010)
$F_m \mid \text{block}, \text{st}_{\text{sd}} \mid C_{\max}$	Takano 等(2019)
$F_m \mid \text{block} \mid \sum C_j$	Fernandez-viagas 等(2016)

本书成稿时，关于 BFS 的研究大部分都集中于目标函数为最大完工时间或总流程时间的问题，并且设定所有待处理任务同时可用(Miyata et al，2019)。虽然最大完工时间和总流程时间这类目标函数便于理论研究，但工业环境中应用较少。在实际调度过程中，任务按照其释放时间实时进入系统中。为了使理论研究更加贴近工业生产环境，本章研究了一个带有释放时间的 BFS 模型，其中优化目标分别为极小化最大完工时间，最大延迟与最大送达时间。从工业意义上讲，极小化最大完工时间可以降低处理器负载进而节省能源，极小化最大延迟旨在满足客户需求进而提高满意度，极小化最大送达时间能够平衡生产成本与物流成本。鉴于这些问题都是强 NP-难的(Lenstra et al，1977)，本章引入了精确算法和智能优化算法求解不同规模的实例，针对小规模实例，使用 B&B 算法，在规定时间内求得最优解，其中精心设计的剪支规则与下界可以剪掉相当多的无效节点。针对中等规模实例，提出了一种混合离散差分进化(hybrid discrete different evolution，HDDE)算法，快速求得近似最优解，其中利用分支定界算法生成的初始种群以及带有插入邻域的局域搜索策略增强了其优化能力。最后，通过数值仿真验证了算法的有效性。

3.2　问题描述与数学模型

在阻塞流水作业调度模型中，n 项不同的任务按照相同的工艺路线经过 m 台串联处理器加工，在任意两台连续的处理器之间不存在缓冲区。即如果下游处理器正在工作，则上游处理器已完工的任务无法进入缓冲区等待，只能继续占用该处理器直至下游处理器空闲为止。工序 $O_{i,j}$ 表示任务 j ($j=1,2,\cdots,n$)需要经过处理器 i ($i=1,2,\cdots,m$)执行。处理时间 $p_{i,j} \geqslant 0$ 表示执行工序 $O_{i,j}$ 花费的时间。任务 j 在释放时间 r_j 进入系统，这是该任务可以开始执行的最早时刻。运送时间 q_j 表示任务 j 完成处理之后交付给客户的时间。交付

日期 d_j 表示任务 j 的交货期或工期，表示事先与客户约定好的交付期限。任务可在交付日期之后完成，但会导致相应的惩罚，如信誉的损失等。完工时间 $C_{i,j}$ 表示工序 $O_{i,j}$ 结束加工的时刻，$C_{m,j}$ 简记为 C_j。送达时间 $Q_j=C_j+q_j$，表示任务 j 交付给客户的时刻。延迟 $L_j=C_j-d_j$，表示任务 j 的延迟时间。每台处理器按照 FCFS 规则执行所有任务，即同顺序处理模式：这些任务按照相同的顺序经过每台处理器。每项任务在处理过程中不允许中断，即某项任务一旦开始处理就要持续至其完工为止。在相同时刻，一台处理器只能执行一项任务，而且一项任务只能由一台处理器加工。优化目标分别为极小化最大完工时间 $C_{\max}=\max\{C_1,C_2,\cdots,C_n\}$，最大延迟 $L_{\max}=\max\{L_1,L_2,\cdots,L_n\}$ 与最大送达时间 $Q_{\max}=\max\{Q_1,Q_2,\cdots,Q_n\}$。

为了建立 MIP 模型，给出如下符号表示。

$S_{i,j}$：工序 $O_{i,j}$ 的开始时间，$i=1,2,\cdots,m;j=1,2,\cdots,n$。

$D_{i,j}$：工序 $O_{i,j}$ 的实际离开时间，$i=1,2,\cdots,m;j=1,2,\cdots,n$。

$x_{k,j}$：0-1 变量。若任务 j 为第 k 个加工的任务，则为 1，否则为 0；$j,k=1,2,\cdots,n$。

Y：足够大的正数。

据此，上述问题可以表示为如下数学规划模型：

$$\text{Minimize } F(C_j),\quad F(C_j)\in\{C_{\max},L_{\max},Q_{\max}\}.$$

$$\text{s.t.}\sum_{k=1}^{n}x_{k,j}=1,j=1,2,\cdots,n \tag{3-1}$$

$$\sum_{j=1}^{n}x_{k,j}=1,k=1,2,\cdots,n \tag{3-2}$$

$$S_{1,j}\geqslant r_j,j=1,2,\cdots,n \tag{3-3}$$

$$S_{i,j}+p_{i,j}=C_{i,j},i=1,2,\cdots,m;j=1,2,\cdots,n \tag{3-4}$$

$$D_{i,j}\geqslant C_{i,j},i=1,2,\cdots,m;j=1,2,\cdots,n \tag{3-5}$$

$$S_{i,j}=D_{i-1,j},i=1,2,\cdots,m;j=1,2,\cdots,n \tag{3-6}$$

$$D_{i,j}\leqslant S_{i,q}+Y\Big(2-x_{k_1,j}-\sum_{k_2>k_1}x_{k_2,q}\Big),$$
$$i=1,2,\cdots,m;j,q,k_1,k_2=1,2,\cdots,n \tag{3-7}$$

$$x_{k,j}\in\{0,1\},r_j\geqslant 0,p_{i,j}\geqslant 0,C_{i,j}\geqslant 0,D_{i,j}\geqslant 0,$$
$$i=1,2,\cdots,m;j=1,2,\cdots,n \tag{3-8}$$

约束(3-1)、约束(3-2)限制了每项任务只能在一个位置加工，且每个位置也只能有一项任务加工。约束(3-3)表示任务在第一台处理器上的开始时间早于其释放时间；约束(3-4)定义了任务的完工时间与其开始时间及处理时间之间

的关系；约束(3-5)限定了任务的离开时间与其完工时间之间的关系；约束(3-6)描述了同一个任务在相邻两台处理器上其开始时间及离开时间之间的关系；约束(3-7)约束了不同任务在同一台处理器上的执行顺序；约束(3-8)定义了相关变量以及参数的取值范围。

3.3　分支定界算法

分支定界是一种用于求解 NP-难问题的隐枚举算法框架，通过系统地搜索状态空间求得小规模问题的最优解。虽然一般的分支定界算法不需要任何加速技术就能得到最优解，但是求解过程却要花费相当长的时间。为了提升分支定界算法的求解效率，常用的加速策略是设计有效的剪支规则和高质量的分支定界算法下界。本章基于任务释放时间的特性及单机最优调度规则，分别提出了剪支规则与分支定界算法下界用于减少有效节点数量，提高算法求解速度。

3.3.1　剪支规则

分支定界算法中剪支规则的作用是尽可能减少无效节点从而减少算法运行时间。设计剪支规则主要借助调度问题中的优化性质。考虑带有释放时间的 BFS 模型中某个可行调度，若将某项已经释放的任务延迟至较晚释放任务完工之后开始处理，则必然会人为地产生较多空闲时间，导致目标函数值恶化。这种情况可以归纳为如下性质。

性质 3.1　对于某个已经固定 $j-1$ 项任务顺序的部分序列，若其紧后任务 $[j]$ 满足如下条件：

$$C_{i,h}(\pi_{j-1}) \leqslant S_{i,[j]} \tag{3-9}$$

则剩余未调度任务的开始时间被延迟，目标函数值恶化。其中，$C_{i,h}(\pi_{j-1})$ 表示任务 h 在位置 j 加工的完工时间，在第 i 台处理器上的完工时间，$i=1,2,\cdots,m$；$j=1,2,\cdots,n$。

显然，不等式(3-9)表明若存在未调度任务 $h\in N'$ 能够在任务 j 开始之前结束其在所有处理器上的加工，则不必将任务 h 安排在任务 j 之后加工，否则将人为增加处理器空闲时间，导致目标函数值变差。若最后一台处理器不满足条件(3-9)，则可以参考如下性质。

性质 3.2　针对 $F_m|\text{block},r_j|C_{\max}$ 问题，若前 $m-1$ 台处理器满足条件(3-9)，则将未调度任务 h 安排在任务 j 之后处理会导致目标函数值恶化。

证明　最后一台处理器的调度问题可以简化为 $1|r_j|C_{\max}$ 问题，其中任务 j 的释放时间定义为 $\hat{r}_j=\max\{C_{m-1,j},C_{j-1}\}$，$j=1,2,\cdots,n$，已知利用 FCFS

规则可以求得其最优解。条件(3-9)表明有 $\hat{r}_h \leqslant \hat{r}_j$，即在处理器 m 上任务 h 的释放时间早于任务 j，由此可以推出结论。

性质 3.3 针对 $F_m|\text{block},r_j|L_{\max}$ 问题，若前 $m-1$ 台处理器满足条件(3-9)且 $d_h \leqslant d_j$，则将未调度任务 h 安排在任务 j 之后处理会导致目标函数值恶化。

证明 最后一台处理器的调度问题可以简化为 $1|r_j|L_{\max}$ 问题，在满足一致性条件的情况下，即 $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$ 且 $r_1 \leqslant r_2 \leqslant \cdots \leqslant r_n$，根据 EDD 规则可以求得最优解。注意到 $\hat{r}_h \leqslant \hat{r}_j$ 且 $d_h \leqslant d_j$，即可推出结论。

性质 3.4 针对 $F_m|\text{block},r_j|Q_{\max}$ 问题，若前 $m-1$ 台处理器满足条件(3-9)且 $q_h \geqslant q_j$，则将未调度任务 h 安排在任务 j 之后处理会导致目标函数值恶化。

证明 最后一台处理器的调度问题可以简化为 $1|r_j|Q_{\max}$ 问题，在满足一致性条件的情况下，即 $q_1 \geqslant q_2 \geqslant \cdots \geqslant q_n$ 且 $r_1 \leqslant r_2 \leqslant \cdots \leqslant r_n$，根据 LDT 规则可以求得最优解。注意到 $\hat{r}_h \leqslant \hat{r}_j$ 且 $q_h \geqslant q_j$，即可推出结论。

在分支过程中，将性质 3.1 分别与性质 3.2～性质 3.4 组合后，形成剪支规则 3.1～规则 3.3 用于求解 $F_m|\text{block},r_j|C_{\max}$、$F_m|\text{block},r_j|L_{\max}$ 和 $F_m|\text{block},r_j|Q_{\max}$ 问题。针对搜索树上的每个候选节点，利用上述剪支规则判断是否需要进行分支。剪支规则剪掉的无效节点越多，分支定界算法的搜索空间就越小，求解效率越高。

3.3.2 分支定界算法下界

对于搜索树中的一条完整分支，在其叶子节点处可以得到一个上界(可行解)。如果新上界目标值更优，则将其更新为当前最好上界(current best upper bound，CBUB)，每当某分支求得的上界更新了当前最好上界之后，就要回溯剪支，删除那些下界值不小于当前最优值的节点。显然，分支定界算法下界越有效、越接近最优解，则剪掉的节点数越多。因此，对于目标函数为极小化的调度问题，设计出尽可能大的下界是十分重要的。考虑某分支节点 $\pi(j)=([1],[2],\cdots,[j])$，$j \in N$，其中前 j 项任务已经固定顺序，这里 N 表示系统中所有任务的集合，$[j]$表示位于第 j 位置的任务。不失一般性，将包含未调度任务的部分序列表示为 $\pi'(j+1)=([j+1],[j+2],\cdots,[n])$。用离开时间 $D_{i,j}$ 表示任务 j 实际离开处理器 i 的时间，则 $D_{i,[j]}=\max\{C_{i,[j]},C_{i+1,[j-1]}\}$，$i=1,2,\cdots,m-1$；$j=1,2,\cdots,n$，当 $i=m$ 时，$D_{i,[j]}=C_{i,[j]}$，工序 $O_{i,h}$ 的最早可用时间 $R_{i,h}$ 可以表示为如下形式，这里 $h \in N'$。

在第 1 台处理器上：

$$R_{1,h}=\max\{D_{1,[j]},r_h\}$$

在第 i 台处理器上：

$$R_{i,h}=\max\{D_{i,[j]},R_{i-1,h}+p_{i-1,h}\}$$

式中，$i=2,3,\cdots,m$。显然，若不考虑任务前后处理关系的约束，则求每台处理器上未调度工序的最优解问题可简化为带有释放时间的单机调度问题。在处理器 i 上，工序 $O_{i,[h]}$ 完工时间的可能估计值为

$$C_{[h]}\geqslant\max_{j+1\leqslant x\leqslant h}\left\{R_{i,[x]}+\sum_{u=x}^{h}p_{i,[u]}\right\}+\sum_{i'=i+1}^{m}p_{i',[h]} \tag{3-10}$$

不等式(3-10)右端项中，除 $\sum_{u=x}^{h}p_{i,[u]}$ 与顺序相关外，其余项都是常数。根据 FCFS 规则可求得 $1|r_j|C_{\max}$ 问题最优解，容易得到分支定界求解 $F_m|\text{block},r_j|C_{\max}$ 问题的分支定界算法下界 LB3.1，其目标值表示如下：

$$Z_{3.1}^{\text{LB}}=\max_{1\leqslant i\leqslant m}\left\{\max_{j+1\leqslant x\leqslant n}\left\{R_{i,[x]}+\sum_{h=x}^{n}p_{i,[h]}^{\text{FCFS}}\right\}+\min_{h'\in N'}\sum_{i'=i+1}^{m}p_{i',[h']}\right\}$$

式中，$p_{i,[h]}^{\text{FCFS}}$ 表示工序 $O_{i,[h]}$ 按照 FCFS 规则进行调度。

根据可中断 EDD(preemptive earliest due date，PEDD)规则可求得 $1|r_j,\text{prmp}|L_{\max}$ 问题最优解，参考不等式(3-10)能够求得工序 $O_{i,[h]}$ 最大延迟的可能估计值为

$$L_{[h]}^{\text{LB}}=\max_{1\leqslant i\leqslant m}\left\{\max_{j+1\leqslant x_1\leqslant x_2\leqslant h}\left\{R_{i,[x_1]}+\sum_{u=x_1}^{x_2}p_{i,[u]}^{\text{PEDD}}-d_{[x_2]}\right\}+\min_{u'\in N'}\sum_{i'=i+1}^{m}p_{i',[u']}\right\}$$

式中，$p_{i,[h]}^{\text{PEDD}}$ 表示工序 $O_{i,[h]}$ 按照 PEDD 规则进行调度。根据最大延迟的定义，容易得到分支定界求解 $F_m|\text{block},r_j|L_{\max}$ 问题的分支定界算法下界 LB3.2，其目标值表示如下：

$$Z_{3.2}^{\text{LB}}=\max\left\{\max_{1\leqslant h_1\leqslant j}L_{[h_1]},\max_{j+1\leqslant h_2\leqslant n}L_{[h_2]}^{\text{LB}}\right\}$$

根据可中断 LDT(preemptive longest delivery time，PLDT)规则可求得 $1|r_j,\text{prmp}|Q_{\max}$ 问题最优解，参考不等式(3-10)能够求得工序 $O_{i,[h]}$ 最大送达时间的可能估计值为

$$Q_{[h]}^{\text{LB}}=\max_{1\leqslant i\leqslant m}\left\{\max_{j+1\leqslant x_1\leqslant x_2\leqslant h}\left\{R_{i,[x_1]}+\sum_{u=x_1}^{x_2}p_{i,[u]}^{\text{PLDT}}+q_{[x_2]}\right\}+\min_{u'\in N'}\sum_{i'=i+1}^{m}p_{i',[u']}\right\}$$

式中，$p_{i,[h]}^{\text{PLDT}}$ 表示工序 $O_{i,[h]}$ 按照 PLDT 规则进行调度。根据最大送达时间的定义，容易得到分支定界求解 $F_m|\text{block},r_j|Q_{\max}$ 问题的分支定界算法下界 LB3.3，其目标值表示如下：

$$Z_{3.3}^{\text{LB}}=\max\left\{\max_{1\leqslant h_1\leqslant j}Q_{[h_1]},\max_{j+1\leqslant h_2\leqslant n}Q_{[h_2]}^{\text{LB}}\right\}$$

3.3.3 算法流程

在给定的搜索树上，分支定界算法沿着每个分支从根节点搜索到叶子节点，在此处能够获得新上界。若新上界值优于 CBUB，则更新 CBUB，并回溯删除无效节点。若所有节点都被删除，则当前 CBUB 就是问题的最优解。否则，重复上述过程，继续搜索具有优势下界值的节点。特别地，如果某节点的下界值等于当前的 CBUB，则直接剪掉该节点。执行搜索过程，直至对全部有效节点完成探索。

为了描述算法方便，给出如下符号定义：Z^{UB} 表示当前最好上界值，$Z_{\tau,j}^{\mathrm{LB}}$ 表示任务 j 在第 τ 层的下界值，$G_{\tau,h}$ 是在第 $\tau-1$ 层搜索树中包括节点 h 全部有效后继节点的集合，其中：$1\leqslant j\leqslant n$；$1\leqslant h$；$\tau\leqslant n$。该分支定界算法的伪代码如伪代码 3-1 所示。

伪代码 3-1 分支定界算法

```
开始
    令 τ=0，N′=N，G_{0,h}=∅，Z^{LB}_{0,j}=0；
    分别采用调度规则 LPTA、EDDA 和 LDTA 生成初始上界 Z^{UB}_0，令 Z^{UB}=Z^{UB}_0
    While
        令 τ=τ+1
        If τ≤n−1
            For j∈N′
                若满足剪支条件，令 Z^{LB}_{0,j}=∞
                If j 未被剪支
                    G_{τ−1,h}=G_{τ−1,h}∪j；
                End if
                计算每个有效节点 j 的下界值 Z^{LB}_{τ,j}
                If Z^{LB}_{τ,j}≥Z^{UB} & j∈G_{τ−1,h}
                    Break
                Else
                    对于所有满足 Z^{LB}_{τ,j}<Z^{UB} 的节点 j，j∈N′，将其按照下界从
                    小到大的顺序加入集合 G_{τ−1}。若存在多个候选节点具有相
                    同的最小下界值，则优先选择任务可用时间早的节点。
                End if
```

```
If G_{τ-1} = ∅
    If τ = 1
        算法终止，当前的上界对应的序列就是最优解
    Else
        τ = τ − 1, N′ = N′ + {π(τ)}
    End if
Else
    选择 G_{τ-1} 中的第一个节点 j，即下界最小的一个节点，将其从 G_{τ-1} 中删除，加入到序列 π 中 π(τ) = j, N′ = N′ − {j}
End if
Else
    计算两个可行解的目标函数值 Z^{UB}_{π(n-1),1} 和 Z^{UB}_{π(n-1),2}
    If Z^{UB} > Z^{UB}_{π(n-1)*} = min{Z^{UB}_{π(n-1),1}, Z^{UB}_{π(n-1),2}}
        Z^{UB} = Z^{UB}_{π(n-1)*}
    用新上界剪支：对于每个层次 1≤τ≤n 的有效节点 j∈G_{τ-1}，若满足 Z^{LB}_{τ,j} < Z^{UB}，则将其从集合 G_{τ-1} 中删除
    End if
End for
End if
End while
结束
```

为了便于理解分支定界算法的求解过程，下面给出具体数值实例。

例 3-1　考虑带有释放时间的阻塞流水作业调度问题，其中包括 3 台处理器$\{M_1, M_2, M_3\}$，4 项任务$\{J_1, J_2, J_3, J_4\}$，目标函数为极小化 $C_{\max}$。任务的释放时间 r_j，处理时间 $p_{i,j}$ 如下所示：

	M_1	M_2	M_3	r_j
J_1	5	4	3	0
J_2	4	6	8	10
J_3	6	5	4	5
J_4	3	2	5	8

使用分支定界算法求解该问题的过程如图 3-1 所示。在第 0 层，采用 LPTA 启发式求得根节点处的初始上界值为 $Z_0^{\mathrm{UB}}=35$。在第 1 层，剪支规则直接剪掉节点 J_2、J_3 和 J_4。在第 2 层，由于节点 J_2 的下界值 $Z^{\mathrm{LB}}=37>Z^{\mathrm{UB}}=$

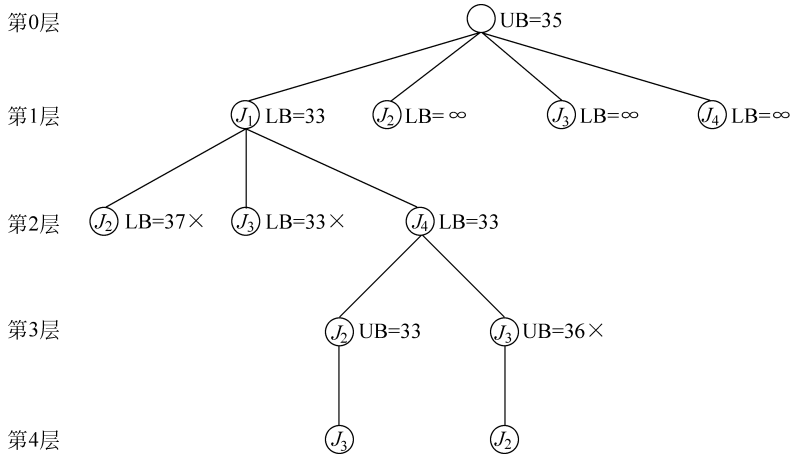

图 3-1 例 3-1 的分支定界搜索树(×表示剪支)

35,于是剪掉该节点;节点 J_3 和节点 J_4 的下界值都是 33,但是由于 $r_4+\sum_{i=1}^{3}p_{i,4}=18<r_3+\sum_{i=1}^{3}p_{i,3}=20$,所以优先选择节点 J_4 进行分支。在第三层时,调度$\{J_1,J_4,J_2,J_3\}$对应的目标函数值是 33,于是更新 $Z^{\mathrm{UB}}=33$,使用新的上界值进行回溯剪支,至此,所有剩余的有效节点都将被剪掉,最优调度为$\{J_1,J_4,J_2,J_3\}$。在此例中,只进行了 4 次计算,大大节省了求解时间。

例 3-2 考虑带有释放时间的阻塞流水作业调度问题,目标函数为极小化 $L_{\max}$。任务的释放时间 r_j,处理时间 $p_{i,j}$ 与例 3-1 相同,交付日期 d_j 如下所示:

	J_1	J_2	J_3	J_4
d_j	16	32	23	28

使用分支定界算法求解该问题的过程如图 3-2 所示。在第 0 层,采用可用最早交付期优先(earliest due date available,EDDA)启发式求得根节点处的初始上界值为 $Z_0^{\mathrm{UB}}=2$。在第 1 层,剪支规则直接剪掉节点 J_2、J_3 和 J_4。在第 2 层,节点 J_2 和节点 J_4 由于下界大于等于当前上界被剪支。在第三层对应两个调度$\{J_1,J_3,J_2,J_4\}$和$\{J_1,J_3,J_4,J_2\}$,目标函数分别为 7 和 2,当前上界没有发生变化。至此,所有剩余的有效节点都被剪掉,最优调度为$\{J_1,J_3,J_4,J_2\}$。同样,此例中只进行了 4 次计算,大大节省了求解时间。

例 3-3 考虑带有释放时间的阻塞流水作业调度问题,目标函数为极小化 $Q_{\max}$。任务的释放时间 r_j,处理时间 $p_{i,j}$ 与例 3-1 相同,运输时间 q_j 如下所示:

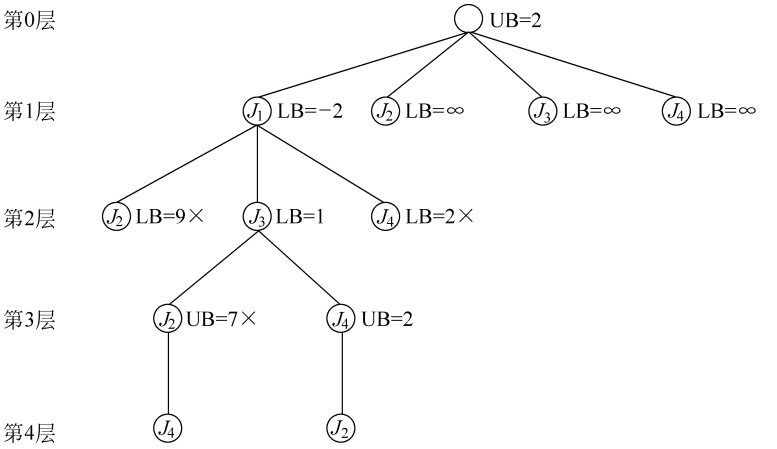

图 3-2　例 3-2 的分支定界搜索树

×表示剪支

	J_1	J_2	J_3	J_4
q_j	6	5	14	8

使用分支定界算法求解该问题的过程如图 3-3 所示。在第 0 层，采用 LDTA 启发式求得根节点处的初始上界值为 $Z_0^{\mathrm{UB}}=39$。在第 1 层，剪支规则直接剪掉节点 J_2、J_3 和 J_4。在第 2 层，节点 J_2 和节点 J_4 由于下界大于等于当前上界被剪支。在第三层对应两个调度$\{J_1,J_3,J_2,J_4\}$和$\{J_1,J_3,J_4,J_2\}$，目标函数分别为 43 和 39，当前上界没有发生变化。至此，所有剩余的有效节点都被剪掉，最优调度为$\{J_1,J_3,J_4,J_2\}$。同样，此例中只进行了 4 次计算，大大节省了求解时间。

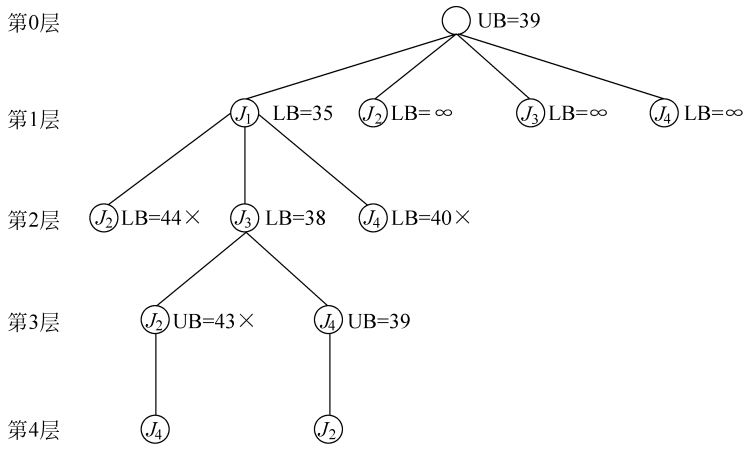

图 3-3　例 3-3 的分支定界搜索树

×表示剪支

3.4 混合离散差分进化算法

虽然分支定界算法能有效地求解小规模问题，但是其本质上是基于枚举的算法，计算时间随着任务数的增加而急剧增长。因此，在实际调度环境中通常使用智能优化算法快速求得近似最优解。DE 算法是一种基于种群进化的随机搜索方法，用于求解复杂连续优化问题。在第 2 章中已经提到经典的 DE 算法中个体采用浮点数编码方式，无法直接用于求解离散优化问题，所以本章采用与第 2 章中一致的编码解码方式，在此不再赘述。下面介绍 HDDE 算法求解阻塞流水作业调度的详细过程。

1. 初始化

通常，HDDE 算法的初始种群是随机生成的。但是，随机解的质量无法保证。因此，在本章设计的 HDDE 算法中，其初始种群中的一部分个体是由分支定界算法生成，即在给定时间内运行分支定界算法，之后采用相应的启发式算法（$C_{\max}$ 对应 LPTA，$L_{\max}$ 对应 EDDA，$Q_{\max}$ 对应 LDTA），将未剪支节点处的不完全序列补全为可行排序作为种群个体；另一部分个体由随机生成的可行排序构成，目的是保证初始种群的多样性，并且要避免两部分种群个体之间的重复。与随机初始种群相比，基于分支定界算法的初始种群删除了无效的子序列，缩小了搜索空间，提高了 HDDE 算法的性能。初始化操作的伪代码如伪代码 3-2 所示。

伪代码 3-2 HDDE 算法初始化

```
输入：运行时长
输出：初始种群
开始
    在给定的时间内运行分支定界算法；
    P0←分支树中的 p 个有效节点的子序列；// P0是一个集合
    在 p 个有效节点的基础上根据启发式规则将未加工任务排序；
    P1←补全的可行解；// P1是一个集合
    P2←根据适应度值将 P1 中的个体排序；// P2是一个集合
    If p＜Popsize do
        Population←集合 P2＋随机生成（Popsize－p）个体
    Else
        Population←P2 中的前 Popsize 个体；
    End if
    Return Population；
结束
```

2. 变异操作

变异操作与 2.5 节中的一致，这里不再赘述，变异操作的伪代码如伪代码 3-3 所示。

伪代码 3-3　HDDE 算法变异过程

输入：Population
输出：V_h^τ
开始
　Do{
　　$a \leftarrow \text{random}(1,\Lambda)$；
　　$b \leftarrow \text{random}(1,\Lambda)$；
　　$c \leftarrow \text{random}(1,\Lambda)$；
　　$d \leftarrow \text{random}(1,\Lambda)$；
　}**While**(a,b,c,d 两两不同)；
　/* 变异操作第一步 */
　For $j=1$ **to** n **do**//n 为任务数
　　$x \leftarrow \text{random}(0,1)$；
　　If $x<\text{MP}$ **do**
　　　$G_{h_1,j}^\tau = X_{a,j}^{\tau-1} - X_{b,j}^{\tau-1}$；$G_{h_1,j}^\tau = X_{c,j}^{\tau-1} - X_{d,j}^{\tau-1}$；
　　Else
　　　$G_{h_1,j}^\tau = 0$；$G_{h_2,j}^\tau = 0$
　　End if
　End for
　/* 变异操作第二步 */
　For $j=1$ **to** n **do**
　　$v_{h,j}^\tau = \text{mod}((x_{\delta,j}^{\tau-1} + (g_{h_1,j}^\tau + g_{h_2,j}^\tau) + n - 1), n) + 1$；
　End for
　Return V_h^τ；
结束

3. 交叉操作

同样地，交叉操作与 2.5 节中一致，这里也不再赘述。交叉操作的伪代码如伪代码 3-4 所示。

伪代码 3-4 HDDE 算法交叉过程

输入：V_h^τ

输出：U_h^τ

```
1  开始
2      For j=1 to n do
3          x←random(0,1);
4          If x≥CP ‖ v_{h,j}^τ in V_h^τ do
5              删除 v_{h,j}^τ ∈ V_h^τ;
6          End if
7      End for
8      U_h^τ ← X_h^{τ-1};
9      从 U_h^τ 移除 V_h 中的任务
10     h←1;
11     While V_h^τ 不为空 do
12         随机地把 V_h^τ 分为 2 个片段;
13         随机地将 2 个片段插入 U_h^τ 中;
14         删除 U_h^τ 中重复的片段
15         h←h+1;
16     End while
17     Return U_h^t
18 结束
```

4. 局域搜索

本节中提出了一种基于迭代贪婪(iterated greedy,IG; Ruiza et al,2007)算法的局域搜索策略用以平衡种群的多样化和集约化。IG 算法是一种基于迭代的启发式算法,具有很强的局部搜索能力,该算法通过破坏和重构来搜索更好的解。在破坏阶段,将候选解 π 分成两个队列：π_D 和 π_R,队列 π_D 由可行解 π 中随机选择的 φ 个任务(称为销毁长度)组成;队列 π_R 则由剩余任务组成。在重构阶段,π_D 中的每一项任务都被插入到队列 π_R 中的最佳位置。基于 IG 算法的局域搜索策略在交叉个体的基础上执行,但是为了节省计算时间,局域搜索以概率 θ 执行。局域搜索策略的伪代码如伪代码 3-5 所示。

伪代码 3-5 HDDE 算法局域搜索

输入：U_h^τ

输出：U_h^τ

```
1   开始
2       h←0;
3       U←U_h^r;
4       π_R←U;
5       While h≤n/5 do
            /* 破坏阶段 */
6           For k=1 to φ do
7               从 π_R 中随机移除一个任务;
8               将任务加入 π_D;
9           End for
            /* 重构阶段 */
10          j←1;
11          While π_D ! =∅ do
12              将 π_D 中第 j 个任务插入 π_R 的最佳位置;
13              j←j+1;
14          End while
15          If f(π_R)<f(U) do
16              U←π_R;
17              h←0;
18          Else
19              h←h+1
20          End if
21      End while
22  结束
```

5. 算法流程

结合以上步骤，HDDE 算法的整个框架伪代码如伪代码 3-6 所示。

伪代码 3-6　HDDE 算法流程

```
    输入：参数 Λ,CP,MP,θ,φ
    输出：bestsofar
1   开始
        /* 初始化阶段 */
```

```
Population←用分支定界算法初始化种群；
计算种群中每个个体的目标函数值；
bestsofar←当前种群中最优的目标函数值；
τ←0；
While 不满足终止条件 do
    For h=1 to Λ do
        /*变异阶段*/
        V_h^τ 基于个体 X_{a,j}^{τ-1}, X_{b,j}^{τ-1}, X_{c,j}^{τ-1} 变异生成；
        /*交叉阶段*/
        U_h^τ ←基于个体 X_h^{τ-1}, V_h^τ 交叉生成；
        /*局域搜索阶段*/
            r ←random(0,1)；
            If r<θ do
                基于局域搜索破坏与重构操作更新个体 U_h^τ；
            End if
        /*选择阶段*/
            If f(U_h^τ)<f(X_h^{τ-1}) do
                X_h^τ=U_h^τ；
            End if
    End for
    更新 bestsofar；
    τ←τ+1；
End while
Return bestsofar；
结束
```

3.5 数值仿真实验

为了验证分支定界算法以及智能优化算法的有效性，本节设计了不同规模的数值仿真实验，用于测试所提出算法的求解性能。所有参与测试的算法采用C++语言编写运行，测试环境为 Intel Core i5-8300 CPU、8GB 内存、Windows 10 操作系统。输入数据 $p_{i,j}$，r_j 和 q_j 的生成方式与 2.5 节中保持一致，在此不再赘述。交付日期 d_j 表示事先与客户约定好的交付期限。任务可在交付日期之后完成，但是会导致相应的惩罚。通常，任务的交付日期要大于其释放时间与处理时间总和。然而，如果每个任务的交付日期都设置得很早或很晚，那么

每个任务的延迟时间分别接近于一个负数或很大的正数，这在实际的调度中是没有意义的。易知，设定交付日期的生成范围时必须保证大多数任务不会在可行的调度中延迟。因此，交付日期 d_j 通过以下方式生成：$d_j=\alpha_j\beta$，其中 α_j 表示通过 FCFS 规则获得的完工时间下界值，$\alpha_j=\min\limits_{1\leqslant j\leqslant n}\left\{r_j+\sum\limits_{i=1}^{m-1}p_{i,j}\right\}+\sum\limits_{j=1}^{n}p_{m,j}$，$\beta$ 由离散均匀分布 $U\left[1-T-\frac{R}{2},1-T+\frac{R}{2}\right]$ 随机生成(Potts et al，1982)。通过一系列数值实验(见附录 B.1)，确定参数 T 和 R 的值分别为 $T=-0.1$，$R=0.4$。测试结果及相关数据展示如下。

3.5.1　分支定界算法

为了突出分支定界算法的优势，这里将其与商业优化软件 IBM® CPLEX 12.8 进行对比，记录各自的运行时间与最终目标值。算法与求解器运行终止时间设定为 10min，若超过 10min，则终止运算记录当前最好目标值。

1. 最大完工时间问题

在测试中，处理器数为 $m=3,5,8$；任务数为 $n=8,10,12,15$。针对不同规模的处理器与任务组合，随机生成 5 组测试算例，采用本章所提出的分支定界算法与 CPLEX 求解器分别进行计算。从表 3-4 中展示的测试结果可以看出，尽管分支定界算法跟 CPLEX 求解器在 600s 内均求得所有测试算例的最优解，但前者在计算效率方面明显优于后者。例如，分支定界算法在 100ms 内求得其中约 63%(19/30)算例的最优解；而 CPLEX 求解器在[100,500]ms 内求得其中约 58%(7/12)算例的最优解。在约 98%(59/60)的算例中，分支定界算法的求解时间都远远小于 CPLEX 求解器，且 CPLEX 求解器的总 CPU 时间约是分支定界算法的 2 倍。可见，分支定界算法的求解效率显著优于 CPLEX 求解器。

2. 最大延迟问题

最大延迟问题的测试算例规模与最大完工时间问题的完全相同。针对不同规模的处理器与任务组合，随机生成 5 组测试算例，采用本章所提出的分支定界算法与 CPLEX 求解器分别进行计算。从表 3-5 中展示的测试结果可以看出，同样地分支定界算法跟 CPLEX 求解器在 600s 内均求得所有测试算例的最优解，但前者在计算效率方面明显优于后者。在约 98%(59/60)的算例中，分支定界算法的求解时间都远远小于 CPLEX 求解器，且 CPLEX 求解器的总 CPU 时间约是分支定界算法的 3 倍。显然，在求解效率上，分支定界算法完全优于 CPLEX 求解器。

表 3-4　最大完工时间问题精确求解对比实验

		$m=3$			$m=5$			$m=8$		
		目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数	目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数	目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数
$n=8$	Trail 1	60	180 ‖ <1	150	90	120 ‖ <1	69	100	210 ‖ 3	365
	Trail 2	76	130 ‖ <1	22	83	110 ‖ 1	163	106	220 ‖ 3	614
	Trail 3	67	140 ‖ <1	152	79	140 ‖ <1	113	99	210 ‖ <1	97
	Trail 4	71	100 ‖ <1	64	82	140 ‖ <1	132	102	190 ‖ 2	704
	Trail 5	65	120 ‖ <1	42	77	150 ‖ 1	71	109	240 ‖ 1	250
$n=10$	Trail 1	80	130 ‖ 1	94	87	180 ‖ 13	3550	118	410 ‖ 13	2076
	Trail 2	79	140 ‖ 1	354	93	210 ‖ 1	258	115	480 ‖ 25	5429
	Trail 3	71	160 ‖ 2	511	94	190 ‖ 2	803	127	230 ‖ 17	2750
	Trail 4	79	160 ‖ 1	671	96	170 ‖ 5	971	118	210 ‖ 23	3711
	Trail 5	63	170 ‖ 5	2763	92	130 ‖ 1	155	116	270 ‖ 5	1490
$n=12$	Trail 1	88	490 ‖ 6	1799	118	2590 ‖ 313	56 159	135	1940 ‖ 207	27 624
	Trail 2	77	510 ‖ 8	2331	103	520 ‖ 37	9293	126	1480 ‖ 262	35 049
	Trail 3	102	340 ‖ 13	7007	103	330 ‖ 145	31 267	126	3860 ‖ 114	13 501
	Trail 4	89	260 ‖ <1	150	121	960 ‖ 88	18 946	124	1470 ‖ 168	21 894
	Trail 5	100	510 ‖ 248	122 431	101	480 ‖ 7	1603	125	1140 ‖ 388	45 782
$n=15$	Trail 1	92	1670 ‖ 11	3110	130	5130 ‖ 579	81719	162	208 230 ‖ 145 204	15 647 812
	Trail 2	109	2670 ‖ 120	52 847	144	1660 ‖ 1151	272 947	156	12 990 ‖ 1446	187 684
	Trail 3	122	1060 ‖ 318	137 226	114	6750 ‖ 331	77 049	152	40 410 ‖ 22 514	2 653 409
	Trail 4	110	790 ‖ 1580	805 605	127	20 700 ‖ 6788	1 091 632	146	53 410 ‖ 20 946	1 997 575
	Trail 5	112	4570 ‖ 704	304 592	126	10 920 ‖ 955	163 988	148	6820 ‖ 2542	314 882

表 3-5　最大延迟问题精确求解对比实验

		$m=3$			$m=5$			$m=8$		
		目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数	目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数	目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数
$n=8$	Trail 1	13	150 ‖ 2	130	12	120 ‖ 0	16	−5	140 ‖ 2	21
	Trail 2	−1	340 ‖ 1	77	24	120 ‖ 1	71	20	150 ‖ 2	71
	Trail 3	3	140 ‖ 2	80	−2	120 ‖ 1	21	2	140 ‖ 5	25
	Trail 4	9	120 ‖ 1	37	10	120 ‖ 1	15	5	180 ‖ 3	16
	Trail 5	9	150 ‖ 1	26	4	140 ‖ 1	45	−6	150 ‖ 1	27
$n=10$	Trail 1	7	210 ‖ 7	332	8	260 ‖ 16	67	38	670 ‖ 130	244
	Trail 2	−7	180 ‖ 3	174	6	230 ‖ 4	118	33	340 ‖ 77	738
	Trail 3	2	200 ‖ 5	104	14	200 ‖ 21	568	14	390 ‖ 137	137
	Trail 4	4	180 ‖ 2	150	24	260 ‖ 23	601	24	540 ‖ 43	582
	Trail 5	−1	200 ‖ 1	88	21	280 ‖ 15	141	−3	530 ‖ 122	612
$n=12$	Trail 1	11	170 ‖ 2	35	1	3280 ‖ 505	48	2	2030 ‖ 647	690
	Trail 2	25	680 ‖ 158	17122	13	600 ‖ 533	3416	11	1030 ‖ 378	1410
	Trail 3	17	370 ‖ 25	1847	−7	750 ‖ 581	110	20	500 ‖ 128	1001
	Trail 4	27	210 ‖ 17	510	−6	590 ‖ 35	1860	−19	760 ‖ 33	342
	Trail 5	5	340 ‖ 27	2543	6	500 ‖ 12	235	1	3870 ‖ 944	5322
$n=15$	Trail 1	−4	2390 ‖ 1826	51 658	13	13 090 ‖ 3627	1867	1	11 560 ‖ 63	44
	Trail 2	−6	1930 ‖ 390	47 475	15	13 290 ‖ 7644	65 428	16	6760 ‖ 595	4130
	Trail 3	−8	1890 ‖ 50	2447	−5	10 730 ‖ 1860	27727	18	29 090 ‖ 2863	1056
	Trail 4	12	1450 ‖ 3896	3773	50	2260 ‖ 11 601	16 868	−16	14 080 ‖ 7558	3608
	Trail 5	−15	3460 ‖ 871	28 940	3	2840 ‖ 690	13 399	19	7890 ‖ 4461	2422

3. 最大送达时间问题

同样地，最大送达时间问题的测试算例规模与最大完工时间问题的完全相同。针对不同规模的处理器与任务组合，随机生成 5 组测试算例，采用本章所提出的分支定界算法与 CPLEX 求解器分别进行计算。从表 3-6 中展示的测试结果可以看出，分支定界算法跟 CPLEX 求解器在 600s 内均求得所有测试算例的最优解，但是在约 97%(29/30)的算例中，分支定界算法求解时间都远远小于 CPLEX 求解器，且 CPLEX 求解器的总 CPU 时间约是分支定界算法的 36 倍。显然，在求解效率上，分支定界算法显著优于 CPLEX 求解器。

3.5.2 混合离散差分进化算法

智能优化算法的寻优性能很大程度上取决于参数的设置。因此，在求解之前需要通过正交实验确定合适的参数(详见附录 B)。任务与处理器的测试规模分别设置为 $m=3,5,10$ 与 $n=60,100,140,180$。对于设定的每种测试规模($m\times n$)，分别生成 10 组测试数据(包括处理时间、释放时间、交付日期、运送时间)。针对每组测试数据，智能优化算法分别进行 5 次重复独立实验。为了验证算法的有效性，分别采用 MRG：$\mathrm{MRG}=\dfrac{Z^{\mathrm{INI}}-Z^{\mathrm{FIN}}}{Z^{\mathrm{FIN}}}\times 100\%$，平均绝对误差(mean absolute gap，MAG)：$\mathrm{MAG}=\dfrac{Z^{\mathrm{INI}}-Z^{\mathrm{FIN}}}{mn}\times 100\%$，其中 Z^{INI} 与 Z^{FIN} 分别表示算法的初始值与最终值；以及 RDP：$\mathrm{RDP}=\dfrac{Z^{H}-Z^{*}}{Z^{*}}\times 100\%$ 作为评价指标，其中 Z^{H} 表示算法求得的最终值，Z^{*} 表示该组测试数据 5 次重复独立实验中所有 Z^{H} 值中的最好值。下面的数据表格中记录了最终结果的平均值。

1. 最大完工时间问题

根据正交实验的结果，HDDE 算法求解问题 3.1 的参数设置为种群规模 $\Lambda=100$，变异概率 MP=0.6，交叉概率 CP=0.2，局域搜索概率 $\theta=0.4$，扰动步长 $\varphi=3$，最大迭代次数 $\tau_{\max}=150$。

表 3-7 中的数据是 HDDE 算法求解问题 3.1 的 MRG 值。实验结果表明，对于给定的处理器规模，随着任务数量的增加，HDDE 算法的优化性能有显著改进。例如，5 台处理器时，当任务数由 60 增加到 180，MRG 值由 6.801%增长到 15.807%。引起该现象的原因可能是问题规模增大引起分支定界生成的初始种群质量下降，导致初始解与最终解之间的误差变大。

表 3-6　最大送达时间问题精确求解对比实验

		$m=3$			$m=5$			$m=8$		
		目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数	目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数	目标函数值	CPU 时间/s CPLEX ‖ B&B	剪支节点数
$n=8$	Trail 1	74	150 ‖ <1	594	99	140 ‖ <1	61	108	200 ‖ 2	227
	Trail 2	87	170 ‖ <1	37	92	210 ‖ 2	512	112	200 ‖ 1	458
	Trail 3	73	200 ‖ <1	144	88	190 ‖ 1	213	106	180 ‖ 1	217
	Trail 4	80	220 ‖ <1	528	90	170 ‖ <1	192	110	220 ‖ 2	594
	Trail 5	73	130 ‖ <1	41	91	150 ‖ 1	397	116	190 ‖ 1	202
$n=10$	Trail 1	91	280 ‖ 1	668	93	370 ‖ 8	2659	123	310 ‖ 7	1170
	Trail 2	84	160 ‖ 1	346	105	270 ‖ 5	1967	126	390 ‖ 12	2994
	Trail 3	82	220 ‖ 3	1285	101	200 ‖ 2	1005	136	210 ‖ 22	2632
	Trail 4	88	180 ‖ <1	460	105	200 ‖ 2	556	133	240 ‖ 27	6318
	Trail 5	68	300 ‖ 10	5820	104	250 ‖ 2	698	130	470 ‖ 8	2724
$n=12$	Trail 1	96	400 ‖ 15	5962	130	1050 ‖ 499	147 182	146	1320 ‖ 344	60 225
	Trail 2	89	380 ‖ 113	68 356	113	450 ‖ 84	25 459	134	1420 ‖ 240	37 348
	Trail 3	112	360 ‖ 14	8569	117	330 ‖ 212	59 571	136	6160 ‖ 228	42 826
	Trail 4	102	400 ‖ 7	3670	133	990 ‖ 180	53 665	129	1040 ‖ 202	34 231
	Trail 5	111	570 ‖ 261	153 188	108	580 ‖ 7	2331	135	750 ‖ 715	100 519
$n=15$	Trail 1	105	940 ‖ 389	115 593	150	5050 ‖ 2756	640 970	171	438 910 ‖ 295 855	54 061 030
	Trail 2	120	4053 ‖ 429	158 348	150	1050 ‖ 25	6466	168	28 740 ‖ 1770	248 898
	Trail 3	132	810 ‖ 223	100 484	126	11 630 ‖ 2813	914 436	164	17 970 ‖ 43 502	7 539 479
	Trail 4	120	1050 ‖ 1038	534 690	135	6280 ‖ 4799	1 028 321	156	100 030 ‖ 27 294	4 019 141
	Trail 5	121	3019 ‖ 901	409 706	132	6810 ‖ 478	98 979	159	5520 ‖ 4794	812 870

表 3-7 HDDE 算法求解问题 3.1 的 MRG 值 %

	$m=3$	$m=5$	$m=10$
$n=60$	6.457	6.801	6.916
$n=100$	8.165	7.132	6.241
$n=140$	8.745	12.370	11.586
$n=180$	11.890	15.807	12.475

为了突出 HDDE 算法的优良性能，表 3-8 记录了 HDDE 与 GA 求解问题 3.1 的对比实验结果。测试中，任务的处理时间取自流水作业调度基准测试集(Taillard，1993)，选取其中的 11 个 $n\times m$ 子集，每个子集包含 10 个实例，这里 $m=5,10,20$ 和 $n=20,50,100,200$。其余参数与 3.5.1 节中生成方式相同。为了进一步减少冗余计算，HDDE 算法使用了改进的终止条件(modified termination condition，MTC)，其中，一旦在 β 次连续迭代之后算法的改进量小于 $\alpha=\dfrac{\delta}{mn}\times 100\%$，则终止迭代，这里 δ 表示两个相邻迭代之间的最小绝对差值。通过一系列的统计检验，最终确定最合适的 $\alpha=2.55\%$，$\beta=6$，即若 HDDE 在连续 6 次迭代后改进量小于 2.55%，则终止计算。GA 的终止时间被设置为 HDDE 的平均终止时间。表 3-8 中，mRDP、minRDP、maxRDP 和 SD 分别表示平均 RDP、最小 RDP、最大 RDP 和标准差。HDDE 算法的 MRDP 值明显优于遗传算法。例如：遗传算法的 mRDP 值的均值为 19.27%，而 HDDE 算法的相应值为 0.76%；而且，前者的 SD 值明显大于后者，这体现了 HDDE 算法的稳定性。

表 3-8 HDDE 与 GA 算法求解问题 3.1 结果比较

$n\times m$	GA				HDDE			
	mRDP	minRDP	maxRDP	SD	mRDP	minRDP	maxRDP	SD
20×5	4.86	1.04	3.22	1.77	0.32	0.00	0.04	0.10
20×10	4.16	1.05	1.90	0.97	1.38	0.00	0.16	0.43
20×20	8.29	0.42	2.79	2.90	0.04	0.00	0.01	0.02
50×5	8.26	3.74	5.87	1.63	0.55	0.00	0.11	0.21
50×10	7.24	2.65	5.00	1.47	0.61	0.00	0.15	0.22
50×20	8.34	1.63	4.76	2.13	0.53	0.00	0.10	0.17
100×5	29.04	18.71	24.06	3.05	0.29	0.00	0.20	0.41
100×10	28.44	16.27	21.79	3.76	1.03	0.00	0.17	0.33

续表

$n\times m$	GA				HDDE			
	mRDP	minRDP	maxRDP	SD	mRDP	minRDP	maxRDP	SD
100×20	21.48	11.73	17.96	3.19	2.06	0.00	0.50	0.75
200×10	49.53	41.63	45.87	2.93	0.67	0.00	0.15	0.26
200×20	42.35	33.40	38.15	2.88	0.88	0.00	0.21	0.30
平均值	19.27	12.03	15.58	2.43	0.76	0.00	0.17	0.29

2. 最大延迟问题

根据正交实验的结果，HDDE 算法求解问题 3.2 的参数设置为种群规模 $\Lambda=100$，变异概率 MP=0.6，交叉概率 CP=0.2，局域搜索概率 $\theta=0.4$，扰动步长 $\varphi=3$，最大迭代次数 $\tau_{\max}=150$。

表 3-9 中的数据是 HDDE 算法求解问题 3.2 的 MAG 值。实验结果显示了与前一节相同的变化趋势，这可能是由相同的原因引起的。然而对于给定的任务规模，随着处理器数量的增加，HDDE 算法的优化性能有所下降。例如：140 个任务时，当处理器数量由 3 增加到 10 时，MAG 值由 33.929%下降到 14.238%。这可能是因为随着处理器数量的增加，每个任务的操作量随之增加，从而增加了相邻任务的空闲时间，降低了最终解的质量。

表 3-9　HDDE 算法求解问题 3.2 的 MAG 值　%

	$m=3$	$m=5$	$m=10$
$n=60$	19.759	18.922	11.061
$n=100$	25.378	21.187	13.323
$n=140$	33.929	21.995	14.238
$n=180$	35.537	32.352	13.146

为了突出 HDDE 算法的优良性能，表 3-10 记录了 HDDE 与 GA 求解问题 3.2 的对比实验结果。测试选取了与上节相同的流水作业调度基准测试集(Taillard,1993)，其余参数与 3.5.1 节中生成方式相同。表 3-10 中实验结果显示的总体趋势与表 3-8 中数据基本一致，这验证了 HDDE 算法在运算性能上要优于遗传算法。

表 3-10 HDDE 与 GA 算法求解问题 3.2 结果比较

$n\times m$	GA				HDDE			
	mADP	minADP	maxADP	SD	mADP	minADP	maxADP	SD
20×5	303.44	200.80	411.20	63.61	5.28	0.00	20.80	7.01
20×10	73.46	29.00	105.60	21.20	3.94	0.00	10.20	3.38
20×20	21.14	10.00	34.90	6.95	1.03	0.00	3.20	1.06
50×5	911.52	531.20	1196.80	205.50	56.40	0.00	113.60	37.13
50×10	207.08	144.60	297.20	48.54	19.64	9.20	42.00	10.75
50×20	50.63	35.55	79.75	14.12	5.14	2.45	8.30	1.61
100×5	1383.12	566.40	2520.80	616.14	52.40	0.00	195.20	66.44
100×10	358.94	237.60	579.40	90.20	37.92	7.60	60.60	17.40
100×20	61.55	42.60	91.15	14.56	5.73	3.60	8.80	1.42
200×10	662.30	456.80	873.40	125.07	55.16	26.00	94.60	16.59
200×20	93.49	69.00	127.85	15.24	7.83	3.85	20.10	4.49
平均值	375.15	211.23	574.37	111.01	22.77	4.79	52.49	15.21

3. 最大送达时间问题

根据正交实验的结果，HDDE 算法求解问题 3.2 的参数设置为种群规模 $\Lambda=100$，变异概率 MP=0.6，交叉概率 CP=0.4，局域搜索概率 $\theta=0.4$，扰动步长 $\varphi=5$，最大迭代次数 $\tau_{\max}=150$。

表 3-11 与表 3-12 中的数据是 HDDE 算法求解问题 3.3 的 MRG 值和 MAG 值。实验结果显示了与前两节相同的变化趋势，对于给定的处理器规模，随着任务数量的增加，HDDE 算法的优化性能有显著改进。例如，5 台处理器时，当任务数由 60 增加到 180 时，MRG 值由 9.542%增长到 28.202%。然而对于给定的任务规模，随着处理器数量的增加，HDDE 算法的优化性能有所下降。又如，180 个任务时，当处理器数量由 3 增加到 10 时，MAG 值由 85.222%下降到 25.522%。上述现象可能是由相同的原因引起的。同时，该表中的数据值略大于表 3-7 与表 3-9 中的值，这说明 HDDE 算法求解问题 3.3 的效率更高。

为了突出 HDDE 算法的优良性能，表 3-13 记录了 HDDE 与 GA 求解问题 3.3 的对比实验结果。测试选取了与上两节相同的流水作业调度基准测试集(Taillard,1993)，其余参数与 3.5.1 节中生成方式相同。表 3-13 中实验结果显示的总体趋势与表 3-8 和表 3-10 中数据基本一致，这验证了 HDDE 算法在运算性能上要优于遗传算法。

表 3-11　HDDE 算法求解问题 3.2 的 MRG 值　%

	$m=3$	$m=5$	$m=10$
$n=60$	8.268	9.542	8.331
$n=100$	9.470	9.854	7.675
$n=140$	9.654	26.051	24.014
$n=180$	27.314	28.202	24.320

表 3-12　HDDE 算法求解问题 3.2 的 MAG 值　%

	$m=3$	$m=5$	$m=10$
$n=60$	19.889	15.600	7.600
$n=100$	23.333	16.720	6.700
$n=140$	22.667	51.029	25.486
$n=180$	85.222	56.422	25.522

表 3-13　HDDE 与 GA 算法求解问题 3.3 结果比较

$n\times m$	GA				HDDE			
	mRDP	minRDP	maxRDP	SD	mRDP	minRDP	maxRDP	SD
20×5	19.388	12.833	28.176	4.161	0.233	0.00	1.169	0.362
20×10	43.509	28.985	54.323	4.937	0.422	0.00	1.408	0.439
20×20	59.978	52.047	65.841	3.369	0.297	0.00	1.240	0.339
50×5	15.612	8.826	23.878	3.316	0.094	0.00	0.629	0.171
50×10	30.04	23.870	40.327	4.004	0.229	0.00	0.707	0.231
50×20	49.201	38.579	54.761	3.769	0.394	0.00	1.244	0.389
100×5	28.642	22.522	39.05	3.791	0.352	0.00	0.993	0.336
100×10	9.076	4.985	13.457	2.008	0.045	0.00	0.332	0.084
100×20	20.528	1.730	27.864	4.239	0.232	0.00	0.951	0.274
200×10	14.571	7.838	19.354	2.748	0.176	0.00	0.736	0.216
200×20	3.306	1.021	6.512	1.586	0.167	0.00	0.560	0.177
平均值	26.714	19.567	33.958	3.448	0.240	0.00	0.906	0.274

3.6　本章小结

本章研究了考虑处理器阻塞的流水作业调度模型，其中优化目标分别为极小化最大完工时间、最大延误与最大送达时间。针对小规模问题，设计了 B&B

算法进行精确求解，针对 3 种问题分别设计了基于 FCFS 规则、PEDDA 规则和 PLDT 规则的分支定界算法下界及有效的剪支策略显著缩小了搜索空间。数值实验证实 B&B 算法的性能明显优于商业求解器 CPLEX。针对中等规模问题，采用混合离散差分进化算法在设定时间内求得高质量可行解。对比实验表明，所提出的算法显著优于其他流行的优化算法。此外，得益于智能优化算法较好的泛化性，本节提出的智能优化算法可以应用与更加复杂的实际工业应用中。