硕 士 学 位 论 文

考虑释放时间的分布式作业调度问题算法研究

Research on Algorithms for Distributed Scheduling Problems with Release Dates

作 者 姓 名：张钰琛

学 科 专 业：管理科学与工程

学　　号：1120200523

指 导 教 师：白丹宇 教授

学 位 类 别：工学硕士

培 养 单 位：大连海事大学

答 辩 时 间：2023 年 6 月

大连海事大学

Dalian Maritime University

摘　　要

随着信息技术与制造业、服务业的深度融合，分布式作业模式应运而生。面对多样化的生产服务需求，企业通过分布式作业来提高自身竞争力。优化分布式作业系统对企业提高效率、节约成本起着至关重要的作用。考虑到任务处理顺序的不同，本文分别研究了固定顺序的流水作业模式与不固定顺序的开放作业模式。

分布式制造是企业不断从供应链角度进行大规模定制运作方式的探索。作为分布式制造的最常见的模式，分布式流水作业的调度优化具有重要意义。本文对分布式流水作业调度问题进行了研究，考虑了任务的释放时间与各个处理单元的作业速度，以极小化最大完工时间为目标来平衡作业负载与效率。对于这个 NP 难问题，建立了混合整数规划模型便于使用优化求解器进行求解。为了更快地获得小规模问题的精确解，提出了一种分支定界算法，设计了性能优异的下界与剪枝规则来加快算法搜索速度；对于中等规模问题，设计了一种离散人工蜂群算法，采用随机可用规则和接受准则策略来提高算法的性能；对于大规模情况，基于问题性质设计了 DLPV-FM 启发式算法，并使其作为分支定界算法的上界来缩小解空间。

分布式服务通过云服务平台打破传统的行业界限，服务项目一般以顺序不定的开放作业模式为主。本文考虑到客户到来的时间与资本介入时的双代理情况，研究了分布式双代理开放作业调度问题，以极小化双代理加权完工时间和为目标，建立了基于位置的混合整数规划模型。对于较大规模的情况，设计了一种有效的 EAF-ADA-DS 启发式算法；对于较小规模的情况，设计了一种离散差分进化算法，通过启发式算法与随机可用初始化来平衡解的质量与多样性。

为了验证所提出算法的有效性，进行了一系列数值实验。对于大规模情况，设计了问题下界与所提出启发式算法进行对比，验证了启发式算法的收敛性；对于中规模情况，将所提出的算法与其他文献所用算法进行对比，证明了设计的智能优化算法的优越性；对于小规模情况，用商业求解器 CPLEX 与设计的分支定界算法进行对比，证明了分支定界算法的高效性能。

关键词：分布式调度；流水作业；开放作业；分支定界；群智能优化算法

Research on Algorithms for Distributed Scheduling Problems with Release Dates

Abstract

The distributed operating mode emerged since the deep integration of information technology with the manufacturing and service industries. To meet diverse production and service needs, companies enhance their competitiveness through distributed operating. Optimizing distributed operating systems plays a crucial role in improving efficiency and saving costs for enterprises. In view of the difference in the order of tasks, this thesis studies two operation modes respectively. One is the distributed flow shop mode with fixed order, the other is the distributed open shop mode with unfixed order.

Distributed manufacturing is an exploration of mass customization operations from the perspective of supply chain. As the most common mode of distributed manufacturing, it is significant to optimizing the scheduling of distributed flow shop. This thesis investigates a distributed premutation flow shop scheduling in which the release date of tasks and the speed of processing units are introduced for simulating real production environment. The goal is to minimize the maximum completion time to balance job load and efficiency. A mathematical programming model is established for handling the NP-hard problem with business optimizer. To obtain exact solutions, a branch and bound algorithm is proposed, where excellent lower bound and pruning rules are designed to speed up the search of the algorithm. For medium scale cases, a discrete artificial bee colony algorithm is designed in which a random available rule and an acceptance criterion are proposed to enhance its performance. For large scale cases, the DLPV-FM heuristic algorithm based on problem properties is designed, and is served as the upper bound of branch and bound algorithm to narrow the solution space.

Distributed service breaks the traditional industry boundaries through cloud service platforms, and the service projects are generally based on the open shop mode with indefinite sequence. This thesis studies a distributed open shop scheduling problem with bi-agent situation and customer arrival time. Aiming at minimizing the sum of weighted completion times by bi-agent, a location-based mixed integer programming model is established. For large scale cases, the EAF-ADA-DS heuristic algorithm is proposed. To achieve satisfied schedules, a discrete differential evolution algorithm is designed, in which the heuristic algorithm and the random available rule are combined to balance the quality and diversity of solutions in initialization.

To verify the effectiveness of the designed algorithms, a serious of numerical simulation experiments are carried out. For large scale cases, the problem lower bounds are designed to

compare with the proposed heuristics, and prove the convergence of the proposed heuristics. For middle scale cases, the proposed intelligent optimization algorithms are compared with those in other literatures, numerical simulation results highlight the superiority of proposed algorithms. For small scale cases, the designed branch and bound algorithm is compared with the commercial solver CPLEX, numerical simulation results demonstrate the excellent performance of the designed branch and bound algorithm.

Key Words: Distributed Scheduling; Flow Shop; Open Shop; Branch and Bound; Population-based Evolutionary Algorithm

目　录

图表目录

图目录

表目录

1 绪论

1.1 研究背景

分布式系统是由多个计算机节点组成的系统，这些计算机节点通过网络互相连接，协同作业，来实现更高效的数据存储、处理和传输等功能。分布式系统具有高度的可扩展性和可靠性，它的多节点架构，使整个系统的容错率大大提升，有效避免了单点故障导致系统宕机的情况。随着计算机技术的不断发展，分布式系统越来越被人们所熟知和使用，并且在各个领域都有广泛的应用。云计算是基于分布式系统的技术，通过将计算、存储和网络等资源进行虚拟化和管理，来实现灵活的资源调配。

近年来，新一轮科技革命和产业革命势如破竹，以网络化、信息化与智能化为核心的第四次工业革命浪潮席卷而来，各行各业和数字技术紧密交融。随着云计算、物联网等先进信息技术与制造业、服务业的深度融合，“云制造”“云服务”等概念被相继提出[1]，突破空间与地域限制的生产经营活动逐渐兴起，各行业在全球范围内通过各类平台进行企业与顾客之间的信息交流，实现企业间的协同和各种社会资源的共享与集成。如何利用这些庞杂的信息合理地分配这些资源，高效、高质量、低成本地为市场提供所需的产品和服务，这是十分重要的调度问题。

工业全球化的趋势对企业的生产制造产生了极大的影响，在全球经济一体化进程中，分布式制造是企业不断从供应链角度进行大规模定制运作方式的探索。企业的制造方式从传统的简单工厂模式逐渐转变为分布式多工厂模式。采用分布式制造模式，企业可以合理配置资源，满足市场的需求，降低成本，提高生产率。

以 3D 打印分布式制造为例，线上建立庞大的 3D 创意模型平台，线下建立遍布全国的分布式制造点，为用户提供 3D 创意设计作品。在整个制作过程中，需要经过建模、切片、打印、组装上色四个阶段，不同的线下制造点的制造速度也会有所差异，对订单在制造点之间的分配，以及对其在制造时的前后顺序会极大地影响订单的完成时间。在订单作业过程中，必须做出两项决定：制造点的订单分配和每个制造点的工作安排。显然，这两个问题是密切相关的，如果想要高效率地完成所有任务，就不能单纯地按照订单到来的顺序进行作业。由这类制造流程简化而成的分布式置换流水作业调度问题，目标是使总完工时间最小，这是在云制造服务中不可避免的问题，并且存在着很大的提升空间。分布式置换流水调度问题也已经成为一个热门的研究领域，是分布式制造的重要发展方向之一。

制造环境的多样性使生产系统变得更加复杂，不同车间的同类设备生产效率也会有很大的差异。同时，不同的加工生产环境、能源质量、工艺流程、机器磨损程度等因素也会影响车间机器运转速率。比如在汽车磨具加工过程中，数控机床会因温差等环境因素存在精度误差，不同工厂会通过数控系统的加减速控制来满足高精度加工要求。而参与钢铁生产的机器的性能则会随着使用时间的增加而退化，设备的磨损会导致输出产品质量的损失。此外，在节能减排的号召下，各行业逐渐开始尝试进行低碳制造，节能要求的不同以及变频控制、数控化等技术的发展，也使得同一企业不同厂区可能采取不同的机器加工计划。

除此之外，随着信息技术由制造向服务的渗透扩展，分布式服务运营也成为了重点发展方向，通过云服务平台打破传统的行业界限，充分调动企业资源的相互协调和聚集。随着科技的迅猛发展，各类云服务平台也不断兴起，企业与客户通过平台来获得各自需要的信息，通过预约、交流等形式实现二者之间线上或者自线上到线下的交互。这类服务一般以分布式开放作业模式为主，如体检、汽车维修等。

在开放作业的实际应用中，客户对于服务项目的选择顺序往往不确定，比如一次普通体检就包括身高、体重、血压、眼耳鼻喉等项目，客户可以以任意顺序进行检查，这种不确定性就增加了整个系统的复杂性。不仅如此，客户选择的服务时间、客户的优先级等因素也会使整个系统的作业调度更加困难。分布式生产服务中，一个很大的难点便是代理介入的问题，生产模式下，存在不同厂商的订单需要在同一时期内处理，服务模式下，存在不同优先级顾客群的任务需要抉择，一个优质的调度方案对于节约服务时间与成本、提高客户满意度具有十分重要的意义。

其中，健康管理平台是云服务平台中应用较多的一类。在通过平台为两个优先级不同的公司提供员工体检预约服务时，存在多个体检点与多个体检项目，各个员工进行体检项目的顺序不受限制，如果不对员工的体检顺序进行排序，会导致体检时间冗长，效率极低，造成时间资源的浪费，因此，需要一个有效的调度方法，在更短的时间里完成两个公司的体检要求，使各方都获得满意的结果。这是典型的分布式双代理开放作业调度问题，在这里，人的身份变成了任务，各个体检点就是每个处理单元，每个体检项目房间便是各个处理机，不同代理之间对资源的竞争会使调度系统更加复杂，调度的最终目标是尽可能最优地满足所有代理方的要求。

1.2 研究意义

分布式作业问题模型可以应用到生产生活的各个方面，对其相关问题模型的探索与优化从未停止，考虑影响作业的实际因素，使其更贴合现实生活，不断更新优化算法，

使其具有更高的性能。分布式生产作为现代工业的一种普遍模式，采用分布式制造模式，可以实现跨地域、跨厂区的资源共享，通过分散的设施网络与信息技术相协调来创造价值，例如富士康和宝马等企业，通过地理上分散的制造设施网络与信息技术协调，进行分散制造来获得更好的收益。分布式服务是分布式运营中的重要一环，采用分布式服务模式，可以对客户群进行分散，作业压力由各个节点分摊，提高效率的同时提高作业水平。

分布式流水作业调度模型与分布式开放作业调度模型是当前分布式制造与服务环境下的两个最常用的作业模型，这两个模型互相补充，分别考虑了流程固定时，作业按照相同顺序被处理的情况，与流程不固定时，作业的处理顺序被打乱的情况。在不同规模的任务量下，为分布式作业设计符合其作业逻辑的高效算法，对于提高分布式作业生产效率，增加行业竞争力与服务水平具有极大意义。

1.3 调度问题概述

调度问题从 20 世纪 60 年代发展至今，已经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学科领域的交叉学科[2]，应用背景也从早期的制造业逐渐延伸到了结合先进技术的非制造领域。调度问题是一类重要的组合最优化问题，它利用一些处理机、机器或资源，最优地完成一系列给定的任务或作业。在执行这些任务或作业时，需要满足一些限制条件，例如作业的到达时间、限定的完工时间、作业的处理顺序、资源对作业时间的影响等。最优地完成是使目标函数达到最小，其中，目标函数通常是对作业时间的长短、处理机的利用率的描述。在调度问题中，处理机的数量和类型，任务或作业的顺序、到达时间、完工限制，资源的种类和性能，都是错综复杂的。并且，在不同的场景中，处理机和任务的角色是可以转换的，例如，在进行手工作业时，人是“处理机”，而在排队候诊时，一个人就是一个“作业”或“任务”。

1.4 调度问题表示方法

调度问题一般由处理机、任务或作业和目标函数这三种要素组成。处理机的种类、数量和环境有多种情况，任务与作业的约束条件也十分复杂，再加上复杂的指标不同的优化目标，形成了丰富的调度问题类型。为了便于阐述，Graham 等[3]提出了三参数表示法来描述不同调度问题的类型，来简化调度问题的表示。

三参数表示法由 $\alpha|\beta|\gamma$ 这三个域组成，以下对本文将会涉及的情况进行介绍。

(1) α 域表示处理机的类型，包括单机、流水作业机器、开放作业机器等，可表示为：

1：单处理机。

P_m：m 个同速机，处理机完全相同。

Q_m：m 个恒速机，处理机不完全相同。

F_m：m 个处理机，流水作业，作业按照相同的顺序经过所有处理机。

O_m：m 个处理机，开放作业，所有作业经过处理机的顺序不受限制。

DF_m：分布式流水作业，存在多个相同的处理单元，每个处理单元都有 m 个处理机，作业按照相同顺序经过所有处理机。

DO_m：分布式开放作业，存在多个相同的处理单元，每个处理单元都有 m 个处理机，所有作业经过处理机的顺序不受限制。

DQ_m：分布式异速流水作业，存在多个处理单元，每个处理单元都有 m 个恒速机，处理机不完全相同，作业按照相同顺序经过所有处理机。

(2) β 域可以包含多项，用以表示作业的限制、要求、性质，资源、环境的种类等约束条件，可表示为：

r_j：任务的释放时间，表示任务在调度系统中的最早可用时间。若 β 域不含这一项，则表示所有任务同时可用。

prmp：表示任务的处理可以在过程中被中断，之后可恢复处理之前被中断的任务。若 β 域没有这一项，则任务一旦开始处理就不可中断。

prmu：同顺序，也称排列或者置换，本约束只适用于流水作业环境。当 β 域含有本约束，则任务按照先进先出(First In First Out, FIFO)原则进行处理，也就是说，所有任务都会按照通过第一个处理机的顺序通过其他的处理机。

nwt：无等待限制只适用于流水作业环境，表示被处理的任务不允许在两个相邻的处理机之间等待，也就是说，所有任务从第一个处理机开始到最后一个处理机中间不会有任何空闲。

block：阻塞现象只适用于流水作业环境。假如在两个处理机之间有个容量有限的缓冲区，当缓冲区满的时候，作业就必须在处理机上等待释放，形成了阻塞。

(3) γ 域用于表明该问题的目标函数，可表示为：

C_{max}：任务的最大完工时间，即以后一项任务离开系统的时刻。优化该目标旨在减少能耗，降低成本。

Σw_jC_j：加权完工时间和，表示所有任务的完工时间（线性加权）总和。若所有任务的权重都相等，则称为完工时间和，记为 ΣC_j。

下面应用上述表示方法描述本文研究的两个问题：

$DQ_m \mid r_j, prmu \mid C_{max}$ 表示考虑释放时间的分布式异速置换流水作业调度问题，目标为极小化最大完工时间。

$DO_m \mid r_j \mid C_{\max}^A + \theta C_{\max}^B$ 表示考虑释放时间的分布式双代理开放作业调度问题，目标为极小化两个代理的加权完工时间和。

1.5 国内外研究现状

本文所研究的 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题与 $DO_m \mid r_j \mid C_{\max}^A + \theta C_{\max}^B$ 问题都是在经典流水与开放调度模型的基础上，充分考虑了分布式、释放时间、双代理等现实情况因素的复杂调度问题。这两类问题受到学者的广泛关注，但是目前的研究成果较少。

1.5.1 分布式置换流水作业调度问题研究现状

流水作业调度是制造业中最常见的优化调度问题模型。置换流水作业调度(Permutation Flowshop Scheduling, PFS) 问题是从现实生产中的流水生产线抽象归纳出来的调度问题。在一个流水作业调度模型中，有 m 台机器和 n 个作业。每个作业需要经过若干操作，每一个操作都要在指定的机器上处理特定的时间，并且经过每个机器的顺序是相同的，其目标是获得一个操作顺序，使给定的目标最优。

对于 $P_2 \mid prmu \mid C_{max}$ 问题，自 Johnson[4]首次提出两机器 PFSP 以及著名的 Johnson 规则以来，该问题逐渐开始受到各界的广泛关注，并成为生产调度的经典问题之一。在作业同时可用的两机的情况下，Johnson 规则可以在 $O(nlogn)$时间内求得最优解，但随着机器数的增加，问题的复杂性发生了变化。Garey 等[5]证明了当机器数大于等于 3 时，PFS 问题是 NP 难的，即该问题不能在多项式时间内获得最优解。而当加入释放时间的约束，问题的复杂性再次变化。Lenstra 等[6]证明了带释放时间的 PFS 问题在两机器时便是强 NP 难的。由于 NP 难问题的计算量会随着问题规模的增加呈现指数增长，所以 NP 难问题在多项式时间内无法求得最优解。因此，通常会应用启发式算法在短时间内获得较好的近似解。Campbell 等[7]在 Johnson 规则的基础上提出了一种解决更通用的 m 机($m \geq 3$)PFS 问题的 CDS 启发式，在计算过程中需要为 m 机 PFS 问题建立 $m-1$ 个辅助问题，通过 Johnson 规则两两比较，在相对较短的时间内求得一个较好的近似解。之后，Nawaz 等[8]又提出了著名的 NEH 启发式算法，该算法运行过程中会在生成初始解后不断选择工件进行插入操作，比较适应度值，并更新适应度最优的排序，Ruiz 等[9]通过实验证明了 NEH 启发式算法的优秀性能。

与单车间调度问题相比，分布式车间调度从单车间演变到了多车间，具有更大的复杂度。分布式置换流水调度 (Distributed Permutation Flowshop Scheduling, DPFS) 问题又称并行流水线问题，需要先将所有待加工的工件分配到各个工厂，然后确定每个工厂内的工件的加工次序。

$DF_m \mid prmu \mid C_{max}$ 问题由 Naderi 和 Ruiz 于 2010 年首次提出[10]，二位学者证明了该问题为 NP 完全问题，并在调度规则、NEH 算法和可变邻域搜索(Variable Neighborhood Search, VNS)算法的基础上，提出了 5 种混合整数线性规划(Mixed Integer Linear Programming, MILP)模型，2 种简单的工厂分配规则，以及 14 种启发式方法、有效的构造启发式方法和变邻域下降算法。由于复杂度与算力的限制，该问题在大规模情况下难以在规定时间内求得最优解，因此通常会采用启发式算法来获得一个满意解。

对于 DPFS 问题，之后的启发式算法多是在 NEH 启发式算法的基础上构造的，学者们会在启发式算法之后加入一些邻域搜索来提高最终解的质量。比如 Gao 和 Chen[11]在 NEH 算法的基础上提出了一个新的构造启发式，该启发式一次向工厂插入一组作业，而不是像原始启发式规则那样一次只插入一个作业，并且该启发式的时间复杂度与原始 NEH 启发式算法的时间复杂度相同。在此基础上，Gao 等[12]利用工厂分配规则又设计了一种改进的变邻域下降(Variable Neighborhood Descent, VND)算法。Pan 等[13]基于 LR[14]和 NEH 算法，分别以分散搜索、离散人工蜂群和迭代贪婪为框架，设计了三种构造启发式。

启发式算法在解决大规模问题时可以在较短的时间内求得一个近似解，但在中规模问题中，启发式算法所得解的精度还是有待提高，这就需要智能优化算法介入，通过智能优化算法优秀的邻域搜索能力，去迭代出更优质的解。

近年来，学者们也更倾向于使用智能优化算法来对不同目标的 DPFS 问题进行进一步的寻优。迭代贪婪(Iterated Greedy, IG)是其中较多被应用于求解 DPFS 问题的一个算法。Hatami 等[15]充分考虑了依赖序列的准备时间对 DPFS 问题的影响，采用 IG 算法并设计了接受准则对其进行求解。Ruiz 等[16]深挖 IG 算法性能，采用嵌套的方法设计了两阶段嵌套的 IG 算法来解决 $DF_m \mid prmu \mid C_{max}$ 问题。Jing 等[17]考虑了到期时间窗的约束，在 IG 算法中增加了一个空闲时间插入评估策略，来最小化总加权提前时间与延误时间。Lu 等[18]优化了一个目标为总能耗与最小化最大完工时间的异构 DPFS 问题。Huang 等[19]设计了一个带有重启策略的 IG 算法来解决考虑基于序列的准备时间的 $DF_m \mid prmu \mid C_{max}$ 问题，Mao 等[20]则考虑了维修情况，设计了一个多起点的 IG 算法。钱斌等[21]以最小化总延误为目标，为 DPFS 问题建立了模型，并提出混合迭代贪婪算法进行求解，基于这个问题的特点提出了最小工期差值规则，并通过 NEH 规则改进算法的初始解。Jing 等[22]优化了邻域搜索策略设计了基于邻域搜索的 IG 算法，首次解决了具有不确定作业时间的 $DF_m \mid prmu \mid C_{max}$ 问题。

人工蜂群(Artificial Bee Colony, ABC)算法在求解 DPFS 问题中也有广泛的应用。Li 等[23]将距离系数纳入考虑范围，使用改进的 ABC 算法进行求解。李浩然等[24]将算法编

码离散化，采用了离散人工蜂群(Discrete Artificial Bee Colony, DABC)算法，解决了异构的 $DF_m|prmu, no\text{-}wait|C_{max}$ 问题。Li 等[25]受分布估计算法(Estimation of the Distribution Algorithm, EDA)启发，在侦查蜂阶段加入 EDA，设计了混合人工蜂群算法(Hybrid Artificial Bee Colony, HABC)算法来解决带有恶化效应的批处理 DPFS 问题。

还有许多基于自然界动物行为的算法被运用于解决 DPFS 问题，比如基于鲸群行为[26, 27]、青蛙行为[28]、灰狼行为[29]与布谷鸟行为[30]智能优化算法。还有一些对于经典算法的进一步创新，如混合遗传算法[31, 32](Hybrid Genetic Algorithm, HGA)和离散差分进化[33](Discrete Differential Evolution, DDE)算法。此外，也有许多其他设计良好的算法被使用，如，EDA [34]，混合免疫算法[35] (Hybrid Immune Algorithm, HIA)， benders 分解算法[36] (Benders Decomposition Algorithm, BDA)，合作协同进化算法[37] (Cooperative Co-Evolutionary Algorithm, CCEA)，合作模因算法[38] (Cooperative Memetic Algorithm, CMA)等。

而针对规模较小的问题，精确算法则表现出了极高的实用性。其中，分支定界算法是求解组合优化问题最常用的精确算法之一，已有学者成功运用分支定界算法来解决置换流水问题。Gmys 等[39]应用分支定界算法求解 $F_m|prmu|C_{max}$ 问题，比较了 45 种分支策略和下界组合。Bai 等[40]则设计了一个序列无关的优质下界来求解 $F_m|prmu, r_j|C_{max}$ 问题。

在现有文献提及的 DPFS 问题模型中，较少考虑释放时间或者都假设作业处理时间相同，不适用于实际大规模数据的仿真测试，在工业意义与实用性方面缺乏参考价值。DPFS 问题发展至今，国内外学者对其进行了深入的思考和研究，根据不同的问题背景，总结出了许许多多的调度模型、调度理论与调度方法，尤其是对于各类元启发式算法的研究越发深入，但是对于以分支定界为代表的精确算法研究上略显粗糙。在现阶段对 DPFS 问题的研究中，除了用求解混合整数线性规划模型的软件在一定规模内可以求出的最优解，并没有其他可以用来精确求解 DPFS 问题的算法。此外，虽然各学者在 DPFS 问题研究中考虑到装配、无等待等条件，但是简化了处理单元作业速度不一致这个约束，极少考虑到不同工厂的性能问题。

本文考虑了实际生产中存在的一些约束，并将这些约束加入了所研究的 DPFS 问题模型。据作者所知，目前的文献中还没有学者对 $DQ_m|r_j, prmu|C_{max}$ 问题进行深入研究，在现有的关于分布式置换调度问题的研究中，大多数假设处理单元的作业速度是完全一致的。并且，释放时间作为生产生活中常见的现实问题，在以往的文献中较少被考虑到。为此，本文研究了一个考虑释放时间和处理单元速度的分布式置换流水作业调度问题，以最大完工时间最小为目标，建立了混合整数规划模型。目前用于求解分布式流水作业

调度问题的方法主要集中在启发式算法和元启发式算法上。对于精确算法的研究较少，本论文根据所提出的模型设计了下界与分支定界算法，用于求得问题的精确解，也便于在小规模情况下测试其他算法的性能。为了在小规模情况下得到精确解，本文设计了针对 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题的剪枝规则与下界，提出了一种基于问题下界的分支定界算法。针对大规模情况，本文提出了动态的最快机器优先最大处理量启发式算法来进行求解，并将其作为分支定界算法的上界来缩小解空间。此外，本文提出了一种离散人工蜂群算法，引入了随机可用规则来平衡解的多样性和质量。同时，在局部搜索中引入接受准则，对离散人工蜂群算法进行了改进。

1.5.2 分布式双代理开放作业调度问题研究现状

传统的开放作业调度问题(Open Shop Scheduling Problem，OSSP)也称为自由作业调度问题，是生产生活中广泛存在的一种作业方式。在一个开放车间调度模型中，有 m 台机器和 n 个作业。每个作业需要经过若干操作，每一个操作都要在指定的机器上处理特定的时间且经过每个机器的顺序不一样，目标是获得一个调度，使给定的目标最优。

相较于流水车间调度和作业车间调度，因为没有工序顺序的特定约束，所以开放作业调度问题灵活性高，具有更大规模的解空间，在有限时间内也更加难以求得最优解。对于所有作业同时可用的情况，在以极小化最大完工时间为目标的开放作业调度问题中，最简单的是两机不带释放时间的开放作业调度问题，用三参数表示法可以表示为 $O_2 \mid\mid C_{max}$，其最优解可以通过最长它机加工时间优先(Longest Alternate Processing Time First, LAPT)规则[41]求得。Gonzalez 等[42]最早开始研究开放作业调度问题，给出了求解 $O_2 \mid\mid C_{max}$ 调度问题的线性时间算法和求解该问题可中断调度的多项式时间算法，并证明了当有两个以上的机器时，以极小化最大完工时间为目标的开放车间作业问题在一般意义上是 NP 难的。稠密排序(Dense Schedule, DS)也是解决开放作业调度问题的高性能启发式，Bárány 等[43]证明，使用 DS 规则获得的最大完工时间最多是最优解的两倍。Chen[44]证明了 $m \geq 2$ 时，DS 的性能比为 $2-1/m$。而当考虑到各个作业的释放时间，即作业不在同一时刻可用，以最小化最大完工时间为目标的开放车间问题可以表示为 $O_m \mid r_j \mid C_{max}$ 问题。Lawer 等[45]指出在两机的时候，即 $O_2 \mid r_j \mid C_{max}$ 问题，是强 NP 难的。Chen 等[46]证明了对于问题 $O_3 \mid r_j \mid C_{max}$，DS 的性能比上限为 2。Bai 和 Tang 对 $O_m \mid r_j \mid C_{max}$ 问题使用 DS 算法进行了渐进性分析，并基于该算法与最短处理时间优先规则设计了一种动态调度算法。其他用来求解开放作业问题的智能优化算法主要还是在经典遗传算法的基础上进行改进。Louis 和 Xu[47]提出了一种参考以往案例的遗传算法；Liaw[48]研究了一种混合遗传算法，将基于禁忌搜索的局部改进过程融入到基本遗传算法中；Rahmani

Hosseinabadi 等[49]设计了一种扩展遗传算法(Extended Genetic Algorithm, EGA)，使用了六种交叉与变异算子，并通过数值实验选择出最有效的算子组合。

经过近四十多年的发展，开放作业调度问题的研究已经十分丰富。双代理作业调度问题作为调度领域中多目标问题的扩展，也备受学者们的关注，越来越多的学者集中研究双代理作业调度中各种组合形式的问题。双代理调度问题不同于 Hoogeveen 等[50]提出的双目标调度问题，在经典的单代理双目标调度问题中，所有的作业都对两个目标产生影响，而在双代理的情况下，某个代理的目标函数只会被该代理作业本身所影响。

学者们的早期研究主要还是在单机或者双机情况下的双代理开放作业调度问题。Agnetis 等[51]研究了两个代理的不同标准组合产生的 9 个问题，在正则函数的最大值（如最大完工时间或延迟）、加权总完成时间和延迟作业的数量三个不同的目标下，解决了在单机与两机情况下，找到单一帕雷托最优解的问题。之后，Agnetis 等[52]还针对两个代理目标不同的三种组合单机双代理调度问题，引入了一种有效的分支定界算法，可以在强多项式的时间内精确地求解。Ng 等[53]研究了双代理单机问题，目标是在给定一个代理最大延迟作业数量的情况下，使另一个代理获得最小加权完工时间。梁建恒等[54]对于单机双代理调度问题，提出了优势代理优先的启发式算法，并证明了其渐进最优性。与这些主要关注调度的学者不同，Mor 和 Mosheiov[55]则研究了两个具有非正则目标函数（目标函数不一定随作业完成时间的增加而增加）的双代理单机调度问题，最小化每个代理作业的最大提前成本或总提前成本，并设计了一种有效的启发式算法进行求解。

针对双代理单机问题的研究，目标函数主要集中在最小化总流程时间[56]、最小化总延误[57]、最小化总加权交期[58,59]等，约束主要考虑了释放时间[60,61]与学习效应[62]等。自 2014 年 Mor 和 Mosheiov[63]研究了双代理流水车间调度问题后，关于多机双代理调度问题的研究逐渐增多。但 Mor 和 Mosheiov 考虑的是双代理流水作业的一种特殊情况，在这种情况下，作业处理时间与机器无关，并且没有考虑到释放时间。同样，Chen 和 Li[64]研究的问题模型中，作业的处理时间也与机器无关。

Jiang 等[65]首次研究了目标为最小化双代理最大加权完工时间和的两机开放车间问题，给出了当代理 B 的权值为任意值时，各代理最大完工时间的加权和最小的复杂性证明，提出了两种基于 LAPT 规则的伪多项式时间算法，在权值为 1 时给出了两种近似算法，在权值大于 1 时给出了另一种近似算法。Jiang 等所研究的是两机上的双代理开放车间问题，对于分布式多机的情况，据作者所知，还没有学者研究。

综上所述，目前对于双代理的研究主要还是集中在双代理单机问题或者双代理流水问题上，并且都假设所有作业同时可用或者假设各个作业的处理时间不受机器的影响。与传统的单处理单元调度问题研究相比，分布式作业调度更符合当今产业全球化发展的

情况，但是分布式作业调度问题研究的问题更为复杂，计算难度大。通过上述文献综述可知，目前对此的研究存在不足，尤其是对于实际情况下的一些额外约束条件上，如不同处理单元之间的处理速度差异，各个阶段作业的处理时间差异等。开放作业调度模型中，任务在各个处理机上处理顺序不受限制，符合体检、检测、维修等实际问题的情况。同时，在实际生活中也需要考虑到代理资本介入的情况。目前对于双代理问题的研究在开放作业调度问题上还是没有较大进展，因此，本文研究了一个考虑释放时间的分布式双代理开放作业调度问题，以极小化双代理完工时间和为目标，建立了基于位置的混合整数规划模型。针对大规模问题提出了有效的启发式算法，并设计了问题下界来验证启发式的性能；针对中规模案例提出了差分进化算法，根据问题特性采用了特殊的编码与解码方式，并在初始化阶段通过随机可用规则与启发式算法平衡了解的质量与多样性。

1.6 主要内容与章节安排

本文以带有释放时间的分布式作业系统为中心，分别针对分布式置换流水作业问题、分布式双代理开放作业问题展开研究，技术路线如图 1.1 所示。本文主要的研究内容与章节安排如下：

第 1 章介绍了论文的选题背景，对调度理论知识和常用的三参数表示法做了较为详尽的说明，对调度问题进行了介绍，最后对国内外的研究现状进行了总结。

第 2 章介绍了调度问题的计算复杂性、不同规模所使用的不同算法以及求解器 CPLEX。

第 3 章研究了 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题。针对该问题，首先建立了混合整数规划模型，设计了最大加工量最快机器优先启发式算法来求解大规模算例，并通过数值实验证明了启发式算法的收敛性。之后设计了精确求解小规模案例的分支定界算法，通过与求解器 CLPEX 的对比实验，证明了分支定界算法的优越性。最后提出了改进的离散人工蜂群算法用于求解中等规模的实例，通过该算法与其他三种算法的数值仿真实验，验证了改进离散人工蜂群算法拥有更好的求解性能。

第 4 章研究了 $DO_m \mid r_j \mid \ C_{\max}^{A} + \theta C_{\max}^{B}$ 问题。针对该问题，首先建立了混合整数规划模型，设计了优势代理优先的稠密启发式算法来求解大规模算例，并通过数值实验证明了启发式算法的收敛性。设计了离散差分进化算法，通过与扩展遗传算法的数值仿真实验，验证了提出的离散差分进化算法拥有更好的求解性能。

第 5 章对本文工作进行了总结与展望。总结了本文对两个调度问题所做的工作，并阐述了对相关问题继续展开深入研究的工作展望。

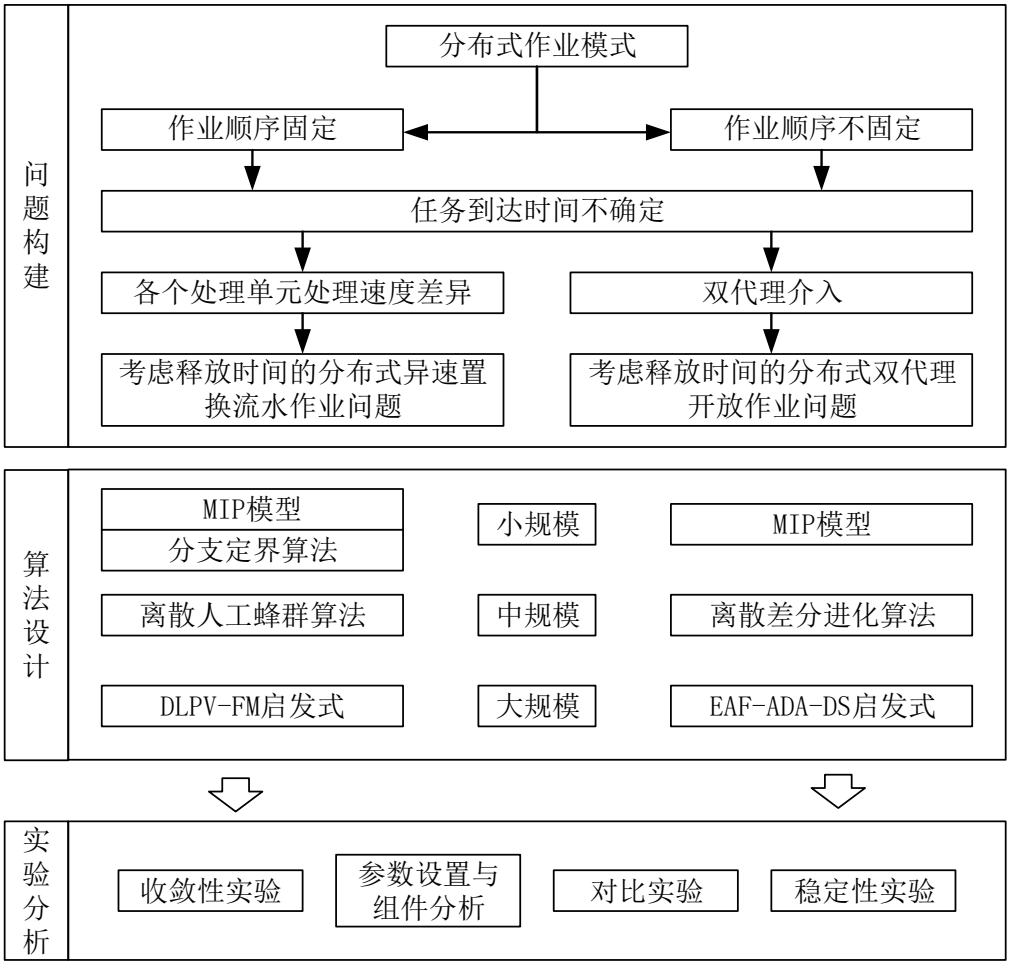


图 1.1　技术路线图

Fig. 1.1　Technology roadmap

2 调度问题复杂性与算法介绍

2.1 调度问题计算复杂性

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，它可以用来评估给定调度问题的难易程度，具体可以根据问题是否能采用多项式时间算法最求得最优解来判断计算复杂性。根据各种调度问题的难度与它们之间的联系，按照复杂性可以分为不同的类。

如果一个判定问题的任何验证都可以在多项式时间内完成，且实例为假时，验证结果总是为假，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。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，可以根据其难度划分为强 NP 难(Strongly NP - hard)和一般 NP 难(Ordinarily NP - hard)两类，其中任何强 NP 难问题不存在伪多项式时间算法。此外，NP 完全(NP - complete)问题指的是一个包含了所有 NP 问题的解法的 NP 问题，因此对于 NP 完全问题，只能采用近似算法或者特定的解法来解决，而不能求解多项式时间内的最优解。

调度问题的计算复杂性对调度问题难易度的判定，直接影响到调度问题的可行性分析和优化研究，同时也影响着优化方法与调度算法的选择，它可以决定算法是采用最优算法还是近似算法。

如果某个调度问题是 P 问题，即存在多项式时间算法，那么主要任务就是尽可能地降低最优算法的时间复杂度，或者在算法的时间复杂度较高时，采用近似算法来求解大规模问题；如果某个调度问题是 NP 难问题，就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。

对于 NP 难的调度问题，在小规模情况下，仍然可以采用精确算法来求得精确解，也可以建立数学模型，通过一些商用优化求解器直接进行求解；对于中规模的情况，可以利用智能优化算法高效的搜索能力，在较短的时间内深入搜索出一个高质量的解；对于大规模的情况，可以利用基于问题规律与经验的启发式算法，在一个令人满意的时间内获得一个较为优质的解。

2.2 调度算法概述

本文研究的问题属于 NP 难问题，难以在多项式时间内求解最优解，需要在不同的规模采用不同的算法进行求解。接下来将根据规模对本文所涉及的算法进行介绍。

2.2.1 精确算法

针对小规模问题，可以利用精确算法进行求解，例如：分支定界(Branch and Bound, B&B)、动态规划等。

B&B 算法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。对于小规模案例，NP 难问题可以用精确算法来解决。是求解 NP 难问题最常用精确算法之一。B&B 算法通常以树形结构枚举解集，丢弃不能产生较好解的无效分支。它包括三个极其重要的步骤：分支、定界和剪支。分支是分解可行解集的过程，就是将一个大问题分解成若干个子问题。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, LB)，而可行解的目标值则称为上界(Upper Bound, UB)。最优解(Optimal Solution, OPT)是解空间中使得目标函数最优的解。三者满足以下关系：$LB \leq OPT \leq UB$。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。当 LB 和 UB 逐渐逼近，直至上下界相等或者解空间全部搜索完毕，得到 OPT。B&B 算法就是通过不断搜寻新的解来更新上界以减小区间，其性能主要取决于分支规则、剪支策略与下界的有效性。

2.2.2 智能优化算法

对于中等规模问题，由于规模的增加使得精确算法难以在允许的时间的求得最优解。因此需要借助智能优化算法来求解调度问题，以求在允许时间内获得一个具有较高性能的可行解。智能优化算法是受自然界中生物进化、物理现象等启发提出的搜索寻优方法。下面介绍一些本文将会涉及到的智能优化算法。

(1) 遗传算法(Genetic Algorithm, GA)：自然界的生物进化是一个不断循环的过程，在这一过程中，生物群体不断地完善和发展。遗传算法就是从生物的进化和遗传中获得了启发，由 Holland[66]于 1975 年提出。GA 根据问题的目标函数构造一个适应度值函数，对一个由多个染色体构成的种群进行评估，通过选择、交叉、变异等操作，在多代繁殖后获得一个适应度值最好的个体作为问题的解。

(2) 人工蜂群(Artificial Bee Colony, ABC)算法：ABC 算法的灵感来自于自然界中蜂群的觅食行为[67]。该算法模拟了蜜蜂的实际采蜜机制，将人工蜂群分为雇佣蜂、跟随蜂和侦查蜂三类，蜂群觅食的蜜源就是待求优化问题的可行解，蜜源的好坏就是当前解的优劣，通过计算适应度值进行比较。雇佣蜂的任务是发现食物源信息并以一定的概率与跟随蜂分享；跟随蜂会根据雇佣蜂分享的食物源信息进行随机选择，然后在被选中的食物源附近进行搜索；在给定的尝试次数之后，如果雇佣蜂与跟随蜂仍未提高解的质量，

则雇佣蜂会变为侦查蜂，其原本拥有的解会被放弃。在达到终止条件后，获得一个质量最好的蜜源作为问题的解。

(3) 迭代贪婪(Iteration Greedy, IG)算法：IG 算法由 Ruiz 和 Stützle[68]提出，该算法主要由邻域搜索、扰动算子和接收准则三个部分组成。在邻域搜索结束后，算法会贪婪地接受更好的解，并以类似于模拟退火的接受准则以一定的概率接受较差的解。之后会对当前解进行破坏与重建，以求跳出局部最优。破坏是对当前解中的元素进行随机抽取，重建是将抽取出的元素重新插入被破坏的解，获得一个完整的新解。IG 算法结构简单，参数较少，在相关调度问题的求解有较好的效果。

(4) 差分进化(Differential Evolution, DE)算法：DE 算法，由 Storn 和 Price 提出[69]，是一种高效的优化算法，可将非线性和不可微的连续空间函数最小化。它通过由总群内各个个体相互协作相互竞争而产生的群体智能来指导优化搜索的方向，主要用于求解实数优化问题，即求解连续型问题。该算法的基本过程是：首先随机产生一个初始种群，从初始种群中选择任意四个个体进行向量做差，与变异系数相比较并生成两个新的个体之后，将这两个新的个体向量相加，再加上当前最好的个体，并进行取模，取模后得到变异个体。对变异个体取概率，与交叉系数相比较，如果大于交叉系数，则删掉对应的基因，如果小于交叉系数，则保留基因。将最后得到的序列插入到选定个体的随机位置，删除非首次出现的基因数，则得到一个新的个体。贪婪地选择，不断地竞争进化保留较为优质的个体，一步步引导搜索朝着最优解逼近。

2.2.3 启发式算法

当面临复杂或者大规模的调度问题时，可以使用启发式算法来解决。启发式算法或者称为调度规则是一种基于经验和模式构造的算法，它根据调度过程中观察到的规律，用经验与知识快速地搜索解，以期达到最优效果。启发式算法构造简单，可以在解决复杂问题时，以较快的速度找到一个满意解，从而节省时间和资源。经典的启发式算法主要有：约翰逊规则、LPT 规则、SPT 规则、NEH 算法等。下面介绍一些本文将会涉及的启发式算法。

(1) LPT 规则：即最长处理时间优先(Longest Processing Time First, LPT)规则。LPT 规则根据作业的处理时间进行排序，优先将处理时间最长的作业分配给处理机，以此来减少任务的完成时间，一般用于以极小化最大完工时间为目标的调度问题。

(2) SPT 规则：即最短处理时间优先(Shortest Processing Time First, SPT)规则。SPT 规则根据作业的处理时间进行排序，优先将处理时间最短的作业分配给处理机，以此来减少任务的完成时间。

(3) DS 算法：即稠密排序(Dense scheduling, DS)启发式算法。DS 算法是一种基于贪心策略的启发式算法，其基本思想是先到先服务。在调度过程中，当任意的处理机出现空闲时，就将此时可用的作业安排到该处理机上处理。对于经典开放作业调度问题，使用 DS 算法可以获得一个不错的解，对于带有释放时间的问题也适应良好。

(4) NEH 算法：NEH 算法[8]是根据其提出者 Nawaz, Enschede 和 Ham 的名字命名的算法，对于置换流水作业问题有着较高的求解效率。NEH 算法基于枚举的思想，将所有作业尽可能插入到最优的位置来有效地减少处理时间。

2.3 调度问题求解器介绍

在对问题建立模型之后，可以通过商业求解器求得小规模情况下的精确解，为了便于模型求解以及算法比较，本文使用求解器 CPLEX 来求解模型。

CPLEX Optimization Studio 由 IBM 公司开发，提供了功能强大且丰富的数学建模工具。CPLEX 是一款全面的优化工具，可以解决复杂的优化问题。CPLEX 为解决调度问题提供了集成编程环境、最优化建模语言以及其他工具，使用了多种优化算法，包括动态规划、整数规划、混合规划、线性规划和非线性规划等。该产品是一个使用户通过数学模型的方式来描述和解决问题的求解引擎，可以完成从数学模型的开发到解决实际问题的所有步骤。它的应用范围非常广泛，可以用于解决资源调度、路径规划、作业调度、排序优化等问题。

CPLEX 使用数学建模语言和专业的优化技术，可以快速而准确地计算出最优解，从而帮助用户更快地达到最佳效果。CPLEX 使用了多种优化算法，可以用于解决资源调度、路径规划、作业调度、排序优化等问题，可以有效提高工作效率。其多种优化算法可以有效提高问题求解的速度，可以解决大规模的线性规划问题，使用户能够在短时间内获得最优解。此外，CPLEX Optimization Studio 还支持多种语言，可以支持 C++、Java、Python 等语言，使得用户可以使用不同的编程语言来建模和解决问题，其支持的多种编程语言可以使得用户更加方便快捷地解决各种问题，使其成为许多行业解决问题的首要选择之一。

3 考虑释放时间的分布式异速置换流水调度问题

3.1 问题介绍与模型构建

分布式流水作业问题模型是分布式制造中最常用的模型，每个任务在被分配至处理单元后，按照特定的顺序经过处理机进行处理。根据现有的生产环境，本文考虑了分布式流水问题中处理单元之间处理速度的差异，在建立问题模型时，对不同处理单元设置了不同的作业速度。为了使问题模型更具有实用性，在模型中引入了释放时间的概念，充分考虑了任务到达时间不一致的情况，使本问题模型更加灵活。

3.1.1 问题描述

在分布式置换流水问题中，包括 $f(f\geq 2)$个处理单元，n 个任务，每个处理单元有 m 台处理机，处理单元之间作业速度不一致。需要将 n 个任务分别分配给 f 个具有不同作业速度 $v_h(v_h>0)$的处理单元，然后确定各个任务在处理单元内的作业顺序。每个任务拥有自己的释放时间和在每一台处理机上的作业时间，并且必须以相同的路径经过 m 台处理机进行作业。每台处理机只能同时处理一个任务，每个任务同时只能由一台处理机作业。任务 j 仅在释放时间 r_j 之后可用。任务 j 需要在处理机 i 上作业的任务量定义为 $p_{i,j}$。$p_{i,j}$ 除以作业速度 v_h，得到处理单元 h 的实际作业时间，$p_{i,j}/v_h$。$S_{i,j}$ 和 $C_{i,j}$ 分别表示在处理机 i 上任务 j 的开始时间和完成时间。假设处理机之间的存储空间足够，即不会产生阻塞，并且当某个处理机开始作业后，在完成之前任务不能被中断。优化目标是找到一个序列，使作业的最大完工时间 $C_{\max}$ 最小，其中，$C_{\max}$ 为所有 $C_{i,j}$ 的最大值。

3.1.2 混合整数规划模型

本节介绍了一个混合整数规划模型用于描述问题。其中所涉及的其他参数与变量如下所示。

参数

n	总任务数
f	总处理单元数
m	总处理机数
l	总位置数
Y	一个极大的正整数
$p_{i,j}$	任务 j 在处理机 i 上的处理量/时间
r_j	任务 j 的释放时间

集合

N 任务集合，索引为 $j\in\{1,\ldots,n\}$

M 处理机集合，索引为 $i\in\{1,\ldots,m\}$

F 处理单元集合，索引为 $h\in\{1,\ldots,f\}$

L 位置集合，索引为 $k\in\{1,\ldots,l\}$

变量

$\lambda_{i,k,j,h}$ =1，如果任务 j 被分配在处理单元 h 中处理机 i 的第 k 位上；=0，否则

$S_{i,j}$ 表示任务 j 在处理机 i 上的开工时间

$C_{i,j}$ 表示任务 j 在处理机 i 上的完工时间

本模型根据任务本身作业时间及其与处理机之间的对应关系建立，通过时间窗的约束使所得解可行，具体如下所示。

$$Minimize\ C_{\max}$$

Subject to:

$$\sum_{k=1}^{l}\sum_{j=1}^{n}\sum_{h=1}^{f}\lambda_{i,k,j,h}=n,i\in M \tag{3.1}$$

$$\sum_{j=1}^{n}\lambda_{i,k,j,h}\le 1,i\in M,k\in L,h\in F \tag{3.2}$$

$$\lambda_{i,k,j,h}=\lambda_{i+1,k,j,h},i\in\{1,\ldots,m-1\},k\in L,j\in N,h\in F \tag{3.3}$$

$$\sum_{j=1}^{n}\sum_{h=1}^{f}\lambda_{i,k,j,h}\le f,i\in M,k\in L \tag{3.4}$$

$$\sum_{k=1}^{l}\sum_{h=1}^{f}\lambda_{i,k,j,h}=1,i\in M,j\in N \tag{3.5}$$

$$\sum_{k=1}^{l}\lambda_{i,k,j,h}\le 1,i\in M,j\in N,h\in F \tag{3.6}$$

$$\sum_{j_1=1}^{n}\lambda_{i,k,j_1,h}-\sum_{j_2=1}^{n}\lambda_{i,k+1,j_2,h}\ge 0,j\in N,k\in\{1,\ldots,p-1\},h\in F \tag{3.7}$$

$$S_{1,j}\ge r_j,j\in N \tag{3.8}$$

$$S_{i+1,j}\ge C_{i,j},i\in\{1,\ldots,m-1\},j\in N \tag{3.9}$$

$$C_{i,j}=S_{i,j}+\sum_{k=1}^{l}\sum_{h=1}^{f}\lambda_{i,k,j,h}p_{i,j}/v_h,i\in M,j\in N \tag{3.10}$$

$$S_{i,j_1}+\lambda_{i,k,j_1,h}p_{i,j_1}/v_h-Y(1-\lambda_{i,k,j_1,h})\le S_{i,j_2}+Y(1-\lambda_{i,k+1,j_2,h}),\quad i\in M,k\in\{1,\ldots,l-1\},h\in F,j_{1,2}\in N,j_1\ne j_2 \tag{3.11}$$

$$S_{i,j}+\lambda_{i,k,j,h}p_{i,j}/v_h-Y(1-\lambda_{i,k,j,h})\le S_{i+1,j}-Y(1-\lambda_{i+1,k,j,h}),\quad i\in\{1,\ldots,m-1\},k\in L,j\in N,h\in F \tag{3.12}$$

$$C_{\max}\ge C_{i,j},i\in M,j\in N \tag{3.13}$$

$$\lambda_{i,k,j,h}\in\{0,1\};C_{i,j}\ge 0;S_{i,j}\ge 0;R_j\ge 0;P_{i,j}\ge 0 \tag{3.14}$$

约束(3.1)确保所有处理单元的任务总数与待处理的任务总数相同；约束(3.2)限制了处理机上每个位置的作业能力；约束(3.3)-(3.6)表示每个任务只能在一个位置和处理单元，并且一个位置最多可以处理一个任务；约束(3.7)防止某个位置被跳过；约束(3.8)确保任务在释放时间之前不能被处理；约束(3.9)表示一个任务在处理机 $i-1$ 上处理后，会到在处理机 i 上处理；约束(3.10)定义了完成时间；约束(3.11)保证在前一个操作完成之前任务不能在同一台处理机上执行；约束(3.12)指出任务在相邻的前一台处理机上完成之前不能被处理；约束(3.13)定义了最大完成时间；约束(3.14)确定了变量和参数的范围。

3.2 DLPV-FM 启发式算法设计与流程

先到先服务(First Come First Serve, FCFS)规则是解决带有释放时间的流水问题的经典启发式算法。但是，FCFS 规则并不能充分表达本问题的特点和目标。当多个任务同时可用时，选择不同的可用任务会对同一处理机上的后续任务存在一定影响。在这种情况下，需要另一个排序规则来使解决方案更优。为此，本文选择了最长处理时间优先(Longest Processing Time First, LPT)规则。在本问题中，每个处理单元有 m 个处理机，即每个任务需要进行 m 个操作，将这 m 个操作看作一个整体，这样，f 个分布式流水处理单元就相当于 f 台并行机，可以用一个任务在所有处理机上的任务量之和作为参数，进行 LPT 规则的排序。经验证，对于不同速度的均匀处理器的一般情况，当 m = 3 时，$C_{\max}(\mathrm{LPT})/C_{\max}(\mathrm{OPT})\le 3/2-1/(2\mathrm{m})$。当 $\mathrm{m}\ge 3$ 时，$C_{\max}(\mathrm{LPT})/C_{\max}(\mathrm{OPT})\le 4/3$ [70]。其中，$C_{\max}(\mathrm{LPT})$表示使用 LPT 规则调度的序列的最大完工时间，$C_{\max}(\mathrm{OPT})$表示最优序列的最大完工时间（可能未知）。因为本问题中任务的作业时间受处理速度的影响，最长处理时间优先规则不够准确。因此，本文提出了最快机器优先作业最大加工量(Largest Processing Volume on the Fastest Machine First, LPV - FM)规则来处理多个任务已经释放的情况。

本文将 FCFS 与 LPV-FM 相结合的启发式算法称为动态的最快机器优先最大处理量(Dynamic Largest Processing Volume on the Fastest Machine First, DLPV - FM)规则，其过程如伪代码 3.1 所示。

伪代码 3.1：**DLPV-FM 算法**

```
1   Begin
2   l←0
3   T←0
4   While
5       If 在当前时间 T 没有可用任务
6           将 T 更新为不可用任务中最早释放的任务 j 的释放时间 r_{i,j}
7       Endif
8       在当前可用任务集中选择任务量最大的作业 j 作为下一个处理的任务
9       计算任务 j 的完工时间并更新当前时间 T
10      If l = n
11          Break
12      Endif
13      l←l+1
14  Endwhile
15  End
```

3.3 分支定界算法

本节将对设计的分支定界算法进行介绍。对于小规模问题，可以使用精确算法来获得最优解。

分支定界算法以树状结构枚举解集，丢弃不能提供更好解的无效分支。该算法由两个核心元素组成：一个是分支规则，它将问题的所有可能解划分为不同的子集；另一个是上下界，计算每个子问题的下界，通过比较上下界的大小将排除那些不太可能产生最优解的子集。分支定界使用上界和下界可以不断缩小解空间，使算法尽快舍弃劣质解。因此，对于本文提出的分支定界算法，在分支定界操作开始时，采用 DLPV-FM 启发式作为初始上界，快速获得一个高质量的可行解。并在充分考虑到 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题性质的基础上，设计了一个优秀的下界，这在分支定界算法的剪支过程中起到了极大的作用。

3.3.1 分支策略

本小节将阐述一系列参考了 Brah 和 Hunsucker[71]的分支策略。为了更直观地描述本文的分支树，本小节将以图 3.1 所示的 2 个处理单元和 4 个任务的分支树为例进行介绍。

可以看到，在第 0 层上的唯一节点是一个虚拟根节点，该节点为整个分支过程的初始上界。分支树中的节点有黑和白两种形式，黑色节点表示从该作业开始的处理单元从

h 切换到了 $h+1$，白色节点表示继续由原处理单元处理。因此，每棵树中的暗节点不能超过 f（f 为工厂的总数），否则处理单元数就会超过问题所定义的数量。为了避免重复操作，每个节点在同一路径上只能出现一次，即每个任务在一条分支上只能被使用一次。第一层包含 n 个黑色节点（n 为任务总数），包含了从每个处理单元出发的所有情况。当 $n \geq f$ 时，如果在某个解中，存在没有处理任务的处理单元，那它很明显是一个劣解。因此，除非任务数量小于处理单元数量，否则不可能存在路径上黑色节点数量小于 f 的情况。

由于处理单元作业速度不同，本问题的分支树不对称，需要对多种情况进行枚举。在第 $n-2$ 层，受到路径中已有黑色节点的影响，会存在两种分支情况。第一种情况是当前分支已有 f 个黑色节点，那么该节点会有两个分支；第二种情况则是该分支的黑色节点少于 f 个，该节点会有如图所示的四个分支。

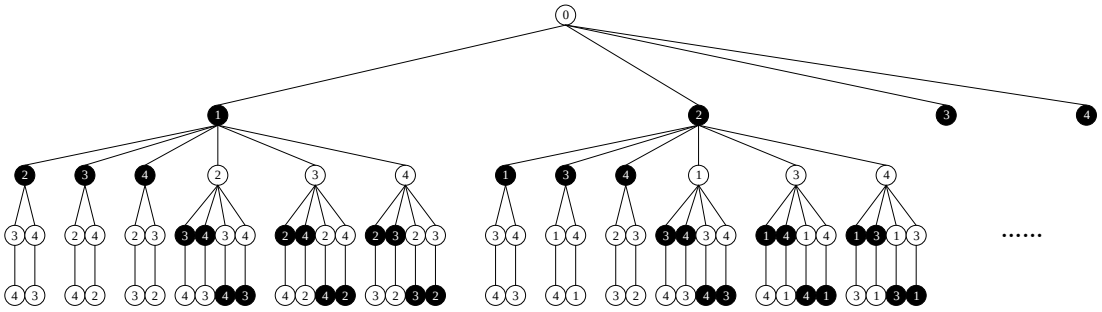


图 3.1　2 处理单元、4 任务分支树图

Fig. 3.1　branch tree diagram of 2 processing unit and 4 tasks

3.3.2　上界设计

分支定界的初始上界是指在解决该问题时，算法较为简单，并且可以快速确定的一个初始的可行解。一个优质的初始上界，能为分支定界剪去大量分支，从而减少运算时间。为了有效地缩小解空间，本文使用前节 3.2 提到的 DLPV - FM 启发式算法来获得可行解，对根节点进行初始化。

3.3.3　下界设计与剪支规则

对于最小化问题 $DQ_m \mid r_j, prmu \mid C_{max}$，只要下界大于或等于上界，后续就可以不考虑该节点，即对其进行剪支。下界需要通过比较每个阶段计算得出的中间值来获得。每个处理单元相同阶段的处理机除了处理速度外相同，因此可以将不同处理单元的同一阶段的处理机看作速度不同的并行机。因为所有的任务都需要在释放时间之后才能执行，所以在假设为并行机的情况下，每个处理机的最早启动时间需要通过分配释放时间最小

的几个任务来决定。空闲时间 I 是在单机情况下计算的，并且是在速度最慢的机器上计算，使空闲时间尽量小，以此来保证下界的可行性。下界公式如下所示：

$$\begin{aligned} LB = \max_{1\le i\le m}\Bigg\{ & \max_{1\le u\le f}\left\{\min_{j\in N'}\left\{r_{i,j,u}^{asce}\right\}, C_{i,u}\right\} + \\ & \left[\sum_{j\in N'} p_{i,j} - \sum_{h=1}^{f}\left(\max_{1\le u\le f}\left\{\min_{j\in N'}\left\{r_{i,j,u}^{asce}\right\}, C_{i,u}\right\} - \max\left\{\min_{j\in N'}\left\{r_{i,j,h}^{asce}\right\}, C_{i,h}\right\}\right)\right] \Big/ \sum_{h=1}^{f} v_h \\ & + \frac{I}{f} + \min_{j\in N'}\left\{\sum_{u=i+1}^{m} p_{u,j} \Big/ v_1^{desc}\right\}\Bigg\} \end{aligned} \tag{3.15}$$

$$I = \max_{j\in N'}\left\{r_{i,j} + \sum_{u\ge j, u\in N'} p_{i,u} \Big/ v_f^{desc}\right\} - \sum_{j\in N'} p_{i,j} \Big/ v_f^{desc} - \max\left\{\min_{j\in N'}\left\{r_{i,j}\right\}, \max_{1\le h\le f}\left\{C_{i,h}\right\}\right\} \tag{3.16}$$

式(3.15)(3.16)中，上标 *asce* 和 *desc* 分别表示升序排序和降序排序。N'表示正在计算的子问题中不包含的任务的集合。

根据问题性质所提出的剪支规则可以表示为：

剪支规则(Pruning Rule, PR)：给定一个节点，其子序列有 $j-1$ 个已分配的任务，如果其紧后任务 j 满足

$$C_{i,j} - p_{i,j} \ge \min_{u\in N'}\left\{\max\left\{r_{i,u}, C_{i,u}\right\} + p_{i,u}\right\}, i = 1...m \tag{3.17}$$

就剪支。

3.3.4 分支定界算法流程

第 0 层上的唯一节点是虚拟根节点，也是初始上界。为了缩小解空间，本文提出的分支定界算法使用前节 3.2 中提到的 DLPV-FM 启发式来获得初始化的可行解来作为分支定界的上界。

为了能够检索更少的子问题，分支定界算法会先搜索完当前层的所有节点，找到下界最小的节点，并将其设置为下一个分支节点。分支定界算法将根据分支策略继续探索分支树，直到获得可行解。如果这个可行解的值大于等于原来的上界，那么就直接返回上一层继续搜索当前下界最小的节点；如果新的上界比原来的上界小，则更新上界，舍弃被上界支配的分支。重复上述过程，直到遍历完所有有效节点。

本文提出的分支定界算法的过程如伪代码 3.2 所示。伪代码 3.2 中使用的字符如下：$LB(l, j)$为第 l 层节点 j 的下界，D_n 为某条路径上的黑色节点数。$G(u, l)$表示节点 u 在第 l 层上所有连续节点的集合。

伪代码 3.2：**分支定界算法**

```
1   Begin
2   l←0
3   用 DLPV-FM 启发式计算初始上界 UB
4   l←l+1
5   While
6       If l≤n−2
7           For j∈N
8               If 节点 u 满足剪支条件 PR
9                   剪去节点 u
10              Else
11                  计算下界 LB(l, j)
12                  If LB(l, j) ≥ UB
13                      剪去节点 u
14                  Endif
15                  选择下界最小的节点 u 作为下一个搜索的节点
16                  l←l + 1
17              Endif
18          Endfor
19      Else
20          If Dn = f
21              计算层 l 中节点 u1 和 u2 的上界 UB(l,u1) and UB(l,u2)
22              If UB > min{UB(l,u1), UB(l,u2)}
23                  UB ← min{UB(l,u1), UB(l,u2)}
24                  剪去满足 UB(l,j) ≥ UB, j∈∪G(u-1,l)的所有节点
25              Endif
26          Elseif Dn ≠ f
27              计算 UB(l,u1), UB(l,u2) , UB(l,u3) , UB(l,u4)
28              If UB > min{UB(l,u1), UB(l,u2), UB(l,u3) , UB(l,u4)}
29                  UB←min{UB(l,u1), UB(l,u2), UB(l,u3) , UB(l,u4)}
30                  剪去满足 UB(l,j) ≥ UB, j∈∪G(u-1,l)的所有节点
31              Endif
32          Endif
33          If ∪G(u - 1, l) ≠ ∅
34              l←l + 1
35          Else
36              Break
```

伪代码 3.2：**分支定界算法**
37 **Endif**
38 **Endif**
39 **Endwhile**
40 **End**

3.4 离散人工蜂群算法

随着案例规模的增长，精确的算法很难在可接受的计算时间内解决优化问题。因此，本节将介绍一种基于 ABC 算法的元启发式算法，以便在更大的规模的问题上获得理想的解。ABC 算法具有全局搜索能力强、收敛速度快的优点，已经很好地解决了一系列组合优化问题[23, 24, 25]。为了更好地适应 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题，本文提出的 ABC 算法采用离散编码，即离散人工蜂群(Discrete Artificial Bee Colony, DABC)算法。

3.4.1 编码与解码

对于 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题，需要确定两个子问题：一是各处理单元的任务分配；二是各处理单元内的任务顺序。本问题在调度过程中，需要同时顾及到处理单元和任务，因此在编码过程中，将使用二维矩阵对其进行编码，行表示各个处理单元，列表示处理单元中任务的调度，即任务的位置。解码过程完全基于编码序列，对任务的处理时间进行累加，其中任务的开始处理时间由它自己的释放时间和处理机的当前时间共同决定。以 2 处理单元、3 处理机和 5 任务为例，编码为如下所示矩阵：

$$\begin{bmatrix} 3 & 1 & \\ 2 & 4 & 5 \end{bmatrix}$$

其任务信息与具体解码情况如表 3.1 和图 3.2 所示。

表 3.1　5 个任务的处理时间与释放时间

Tab. 3.1　processing times and release date of 5 tasks

	J_1	J_2	J_3	J_4	J_5
M_1	3	1	5	4	2
M_2	2	3	2	2	2
M_3	4	1	3	1	2
r	7	2	0	4	7

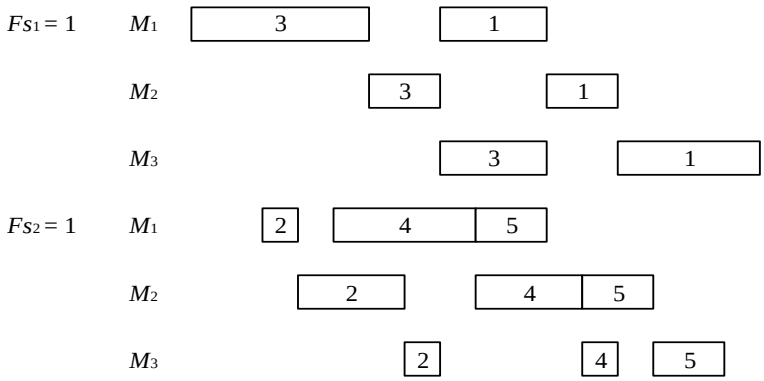


图 3.2　2 处理单元、3 处理机和 5 任务的解码结果

Fig. 3.2　decoding result for 2 processing unit, 3 machines and 5 tasks

3.4.2　初始化策略

为了平衡初始解的质量和多样性，根据问题的性质引入随机可用(Random Available, RA)规则。RA 规则就是在当前已释放的任务中进行随机选择，初始化过程如伪代码 3.3 所示。

伪代码 3.3: **InitializationRA**

```
1  Input: 序列 π
2  Output: π
3  Begin
4  T ← 0
5  For l = 1 to n do
6      If 如果在当前时间 T 没有已释放的可用任务
7          将 T 更新为未释放任务集合中最小的释放时间
8      Endif
9      在可用任务集合中随机选择一个任务 j 作为下一个处理的任务
10     计算任务 j 的结束时间并更新当前时间 T
11 Endfor
12 End
```

3.4.3　接受准则

在邻域搜索过程中如果采用完全贪婪的接受准则可能会错失一些优化的可能性。为了保证 DABC 算法的探索性能，在邻域搜索过程中增加了基于运行时间的接受准则。如果是更好的解，那就接受它；而当较差的解满足下列公式时，也将接受它

$$\beta \cdot \theta < \frac{totalT - curT}{totalT} \tag{3.18}$$

否则，较差的解将被舍弃。式(3.18)中 θ = rand[0,1)， β 是用来调整公式的常数参数，*totalT* 为 CPU 总运行时间，*curT* 为 CPU 当前运行时间。

3.4.4 邻域搜索策略

在邻域搜索部分，本算法使用了三种邻域搜索方法，包括精心设计的位移算子 ShiftRA 和两个常用的交换算子 Swap1 和 Swap2。

ShiftRA：从完工时间最大的关键工厂中随机抽取一个任务 *j*，然后随机选择一个目标工厂，根据 RA 规则将任务 *j* 插入目标工厂。

Swap1：从关键工厂和另一个随机选择的工厂中，随机抽取两个作业 *i, j*，并交换它们的位置。

Swap2：从一个随机选择的工厂中，随机抽取两个作业 *i, j*，并交换它们的位置。

在执行邻域搜索操作时，将选择上述算子中的一个来搜索邻域解。如果新解的适应度值小于原解，则接受新解；如果新的调度方案较差，但满足接受标准，那么新解也将被接受，否则将维持原解。其过程如伪代码 3.4 所示。

伪代码 3.4：**Local search**

```
1  Input: 序列 π
2  Output: π
3  Begin
4  从下列三个算子中随机选择一个进行邻域搜索
5  ShiftRA(π) ← π'
6  Swap1(π) ← π'
7  Swap2(π) ← π'
8  If f(π') < f(π)
9        π' ← π
10 Elseif f(π') ≥ f(π)
11       θ ← rand(0, 1)
12       If  β·θ < (totalT − curT)/totalT
13             π' ← π
14       Endif
15 Endif
16 End
```

3.4.5 算法阶段设计

(1) 雇佣蜂阶段

雇佣蜂的任务是发现食物源信息并以一定的概率与跟随蜂分享。雇佣蜂的数量与种群规模 *Pop* 一致，对于每个食物源，雇佣蜂都会找个新相邻食物源，并以设计的接受准则选取食物源。其过程如伪代码 3.5 所示。

伪代码 3.5：**雇佣蜂阶段**

```
1  Input: 种群集合 PS {π1, π2, …, πPop},未更新次数集合 Tlimit(), 全局最优解 πbest
2  Output: PS {π1, π2, …, πPop}, Tlimit(), πbest
3  Begin
4  For l = 1 to Pop do
5      Local search(πl) ← πl
6      If f(πi) < f(πbest)
7          πbest ← πl
8      Else
9          Tlimit(πl) ← Tlimit(πl) + 1
10     Endif
11 Endfor
12 End
```

(2) 跟随蜂阶段

跟随蜂会根据雇佣蜂分享的食物源信息进行随机选择，然后在被选中的食物源附近进行搜索。为了使搜索到富裕食物源的可能性更大，在跟随蜂阶段采用轮盘赌的选择方法。每个蜜源被选中的概率为其本身适应度值的倒数与所有蜜源总和的比值，为了避免倒数太小，在计算时会适当对其加倍。其过程如伪代码 3.6 所示。

伪代码 3.6：**跟随蜂阶段**

```
1  Input: 种群集合 PS {π1, π2, …, πPop},未更新次数集合 Tlimit(), 全局最优解 πbest
2  Output: PS {π1, π2, …, πPop}, Tlimit(), πbest
3  Begin
4  For l = 1 to Pop do
5      根据轮盘赌原则从 PS 中选择一个解 πx
6      Local search(πx) ← πx
7      If f(πi) < f(πbest)
8          πbest ← πl
9      Else
```

伪代码 3.6：**跟随蜂阶段**

```
10            T_limit(π_x) ← T_limit(π_x) + 1
11        Endif
12    Endfor
13    End
```

(3) 侦查蜂阶段

在给定的尝试次数之后，如果雇佣蜂与跟随蜂仍未提高解的质量，则雇佣蜂会变为侦查蜂，其原本拥有的解会被放弃。这个尝试次数是事先给定的放弃阈值(Abandonment Criteria, AC），一旦超出该值，侦查蜂会开始重新搜索一个新的解。为避免重复，在侦查蜂阶段不再使用之前提出的带 RA 规则的初始化，而是对原解进行大规模的破坏与插入。在原解中随机抽取 N/2 个工件，对于每个被抽取的工件，找到剩余解中该工件已释放的位置，对其进行前插操作。其过程如伪代码 3.7 所示。

伪代码 3.7：**侦查蜂阶段**

```
1   Input: 种群集合 PS {π_1, π_2, …, π_Pop}, 未更新次数集合 T_limit(π), 放弃阈值 AC, 全局最优解 π_best
2   Output: PS, π_best
3   Begin
4   For i = 1 to Pop do
5       If T_limit(π) ≥ AC
6           For j = 1 to n / 2 do
7               从子序列 π 中随机选择一个任务并将其存入子序列 π_sub
8           Endfor
9           For j = 1 to n / 2 do
10              根据存入顺序从子序列 π_sub 中抽出任务 j，将任务 j 插入子序列 π 中第一个开
                始处理时间比任务 j 的释放时间晚的任务之前
11          Endfor
12          If f(π_i) < f(π_best)
13              π_best ← π
14          Endif
15      Endif
16  Endfor
17  End
```

3.4.6 离散人工蜂群算法流程

DABC 算法的过程如伪代码 3.8 所示，其中 PS 为种群集合，π_{best} 表示全局最优解，$T_{limit}(\pi)$表示调度未更新次数集合。

伪代码 3.8：**DABC 算法**

```
1  Begin
2  创建初始种群集合 PS {π1, π2, …, πPop}
3  For i = 1 to Pop do
4      InitializationRA(πi) ← πi
5      If f(πi) < f(πbest)
6          πbest ← πi
7      Endif
8  Endfor
9  tatalT ← m·n·f·50
10 curT ← 0
11 Tlimit() ← { 0 }
12 While curT ≤ totalT do
13     雇佣蜂阶段(PS, Tlimit(), πbest) → PS, Tlimit(), πbest
14     跟随蜂阶段(PS, Tlimit(), πbest) → PS, Tlimit(), πbest
15     侦查蜂阶段(PS, Tlimit(πi), AC, πbest) → PS, πbest
16 Endwhile
17 End
```

3.5 数值仿真实验

为了验证本文所提出的各个算法的有效性，本节采用数值仿真实验对其进行测试，LPTA、B&B 和 DABC 算法都使用 Visual Studio 软件用 C++进行编写，运行于 Intel(R) Core(TM) i5-9400 CPU 2.90GHz，内存 8GB 的 Windows10 操作系统上。

3.5.1 分支定界算法数值实验

为验证所设计的分支定界算法的优越性，本节用求解器 CPLEX 12.8 与分支定界算法进行对比实验。在数值实验中处理机数与任务数分别为：m = {2,4,6}, n = {8,10,12}。处理机数与任务数两两配对组合，共 9 组，每种组合有 10 个案例。由于计算机能力的限制，处理单元数量保持在 2。经测试，在 n = 12 时，CPLEX 已经不能在 3 小时内求得解，而分支定界仍然能在十分钟内获得解，因此考虑到 CPLEX 不能在一定时间内获得精确解的情况，为便于比较，本文将其数据记录为十分钟时获得的可行解。

从表 3.2 可以看出，在规模较小的情况下，分支定界算法和 CPLEX 算法都能得到最优解，而分支定界算法的运行时间明显比 CPLEX 短很多。随着规模的扩大，CPLEX 逐渐吃力，难以获得最优解，特别是当任务数量增加到 12 个时，它已经不再能在 3 小时内获得最优解，而分支定界算法仍然可以获得最优解。可以看出，本文提出的分支定界算法是有效的。剪支规则在缩小解空间和节省计算时间方面有很大的助益，比如在 $f=2, m=4, n=10$ 的第一个案例中，该剪支策略直接切割了 142907 个节点，搜索了 100564 个$(142907/100564\approx 142.11\%)$个节点。现有的结果清楚地证明了所提出的分支定界算法的出色性能，体现出所设计的分支定界算法在小规模案例中对求解器的支配地位。

3.5.2 DLPV-FM 启发式算法数值实验

为了验证 DLPV-FM 算法作为大规模情况下启发式算法与分支定界算法上界的性能，本小节设计了详尽的数值实验如下：测试了 $f=\{2,3,5\}$，$m=\{3,5,7\}$的 9 种不同组合，每个组合随机生成 20 组案例，分别在 $n=\{50, 100, 200, 500, 1000, 2000\}$的规模下运行。其中的一半处理单元，处理速度简单地设置为 2，而对于其他处理单元，处理速度设置为 1。为了更接近实际生产，在释放后可以直接处理的任务和需要等待的任务的比例为 1/3 到 1。生成的释放时间均匀分布在$[0,\ dip\cdot m\cdot n/f\ ]$区间内，其中 $dip=[5, 14]$根据规模确定。

$DQ_m \mid r_j, prmu \mid C_{max}$ 问题下界的计算和分支定界算法下界的计算之间有一些区别，算法下界需要计算部分序列的情况，而问题下界直接计算完整序列。为适应 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题，问题下界公式设计如下：

$$LB_{pro}=\max_{1\le i\le m}\left\{r^{asce}_{i,j,f}+\left[\sum_{j=1}^{n}p_{i,j}-\sum_{h=1}^{f}\left(r^{asce}_{i,j,f}-r^{asce}_{i,j,h}\right)\times v_h\right]\Big/\sum_{h=1}^{f}v_h+\frac{I_{pro}}{f}+\min_{1\le j\le n}\left\{\sum_{u=i+1}^{m}p_{u,j}\Big/v^{desc}_1\right\}\right\} \tag{3.19}$$

$$I_{pro}=\max_{1\le j\le n}\left\{r_{i,j}+\sum_{u=j}^{n}p_{i,u}\Big/v^{desc}_f\right\}-\sum_{j=1}^{n}p_{i,j}\Big/v^{desc}_f-\min_{1\le j\le n}\left\{r_{i,j}\right\} \tag{3.20}$$

对每一组案例进行计算之后，取相同规模下 20 组案例所得结果的平均值，与对应的下界结果的平均值进行比较，其结果如图 3.3 所示。可以看到，随着规模的逐渐增大，二者的差距逐渐减小，这意味着在规模增大的情况下，上界和下界逐渐相互逼近。从趋势线可以看出，DLPV-FM 启发式在解决 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题时具有收敛性。

表 3.2　分支定界算法与 CPLEX 对比结果

Tab. 3.2　Computational results between B&B and CPLEX

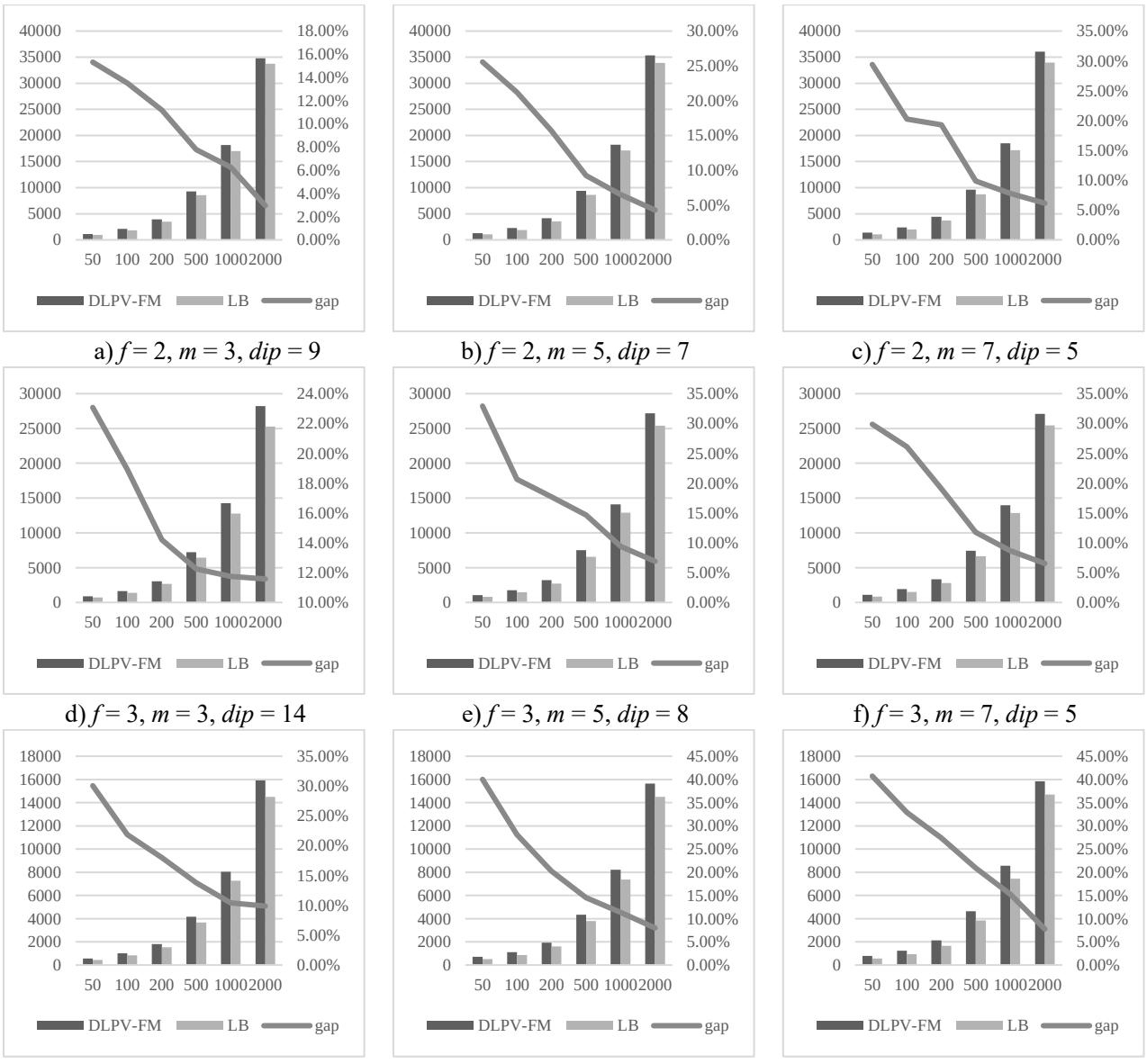
Scale *f - m - n*	**case**	**Value**		**CPU time/ms**	
		B&B	**cplex**	**B&B**	**cplex**
2-2-8	0	208	208	19	10065
	1	186	186	33	7029
	2	170	170	9	2070
	3	185	185	37	5011
	4	162	162	29	7072
	5	183.5	183.5	51	10047
	6	174	174	47	4075
	7	207	207	27	6026
	8	194.5	194.5	54	3091
	9	247.5	247.5	65	7036
2-4-8	0	285	285	38	19096
	1	239	239	34	10005
	2	232	232	105	9089
	3	252.5	252.5	165	13058
	4	293.5	293.5	423	15034
	5	267.5	267.5	210	8019
	6	268	268	202	14088
	7	253.5	253.5	291	10093
	8	260	260	164	9098
	9	282	282	426	18031
2-6-8	0	340	340	555	23028
	1	294	295	332	19067
	2	321.5	321.5	633	32019
	3	315	315	354	23053
	4	352.5	352.5	625	18061
	5	348	348	712	17083
	6	326	326	453	17009
	7	334	334	251	21058
	8	308	308	416	17073
	9	306	306	457	21003

表 3.2　分支定界算法与 CPLEX 对比结果（续）

Scale *f - m - n*	case	Value		CPU time/ms	
		B&B	**cplex**	**B&B**	**cplex**
2-2-10	0	233	233	1184	197041
	1	234	234	1561	112077
	2	232.5	232.5	3917	--
	3	206	206	798	161040
	4	225.5	225.5	12334	427083
	5	208	208	1465	149031
	6	235	235	569	542010
	7	242	242	2575	355070
	8	247.5	247.5	7610	243034
	9	227	227	4707	109027
2-4-10	0	291.5	291.5	3979	478069
	1	265	265	9874	456016
	2	313	313	6382	--
	3	332	332	16015	--
	4	327	327	7936	253064
	5	**317**	317.5	5689	--
	6	302.5	302.5	19681	--
	7	283	283	10015	123031
	8	311	311	11151	468039
	9	289.5	289.5	36085	--
2-6-10	0	367.5	367.5	10658	--
	1	350.5	350.5	72428	--
	2	392	392	9064	--
	3	422	422	6485	352029
	4	369	369	63095	482061
	5	**373.5**	382	15771	--
	6	415	415	19914	--
	7	**343**	345.5	14538	--
	8	**372.5**	390	17366	--
	9	**431.5**	438	63278	--

表 3.2　分支定界算法与 CPLEX 对比结果（续）

Scale *f* - *m* - *n*	case	Value		CPU time/ms	
		B&B	**cplex**	**B&B**	**cplex**
2-2-12	0	**281**	282	19138	--
	1	**235**	236.5	6968	--
	2	**226**	228	48571	--
	3	**256**	257.5	168495	--
	4	**230**	231.5	14804	--
	5	205.5	205.5	18964	--
	6	**302**	307	8142	--
	7	**278.5**	279	47859	--
	8	249	249	6408	--
	9	**256**	257.5	8416	--
2-4-12	0	**357**	379	69741	--
	1	302	302	660745	--
	2	**343**	350.5	443719	--
	3	**364**	376	207049	--
	4	**354.5**	374	266419	--
	5	**375**	381	260483	--
	6	**344**	354	565504	--
	7	**338.5**	348	363231	--
	8	**313.5**	340	311566	--
	9	**326**	338.5	490902	--
2-6-12	0	**411**	423	3496205	--
	1	**377**	422	612848	--
	2	**437.5**	454	4687988	--
	3	**437.5**	440	1070960	--
	4	**426**	449	176679	--
	5	**436.5**	458.5	172440	--
	6	**410.5**	414	434200	--
	7	**413.5**	436	949432	--
	8	**428**	451	2476255	--
	9	**390**	429	914475	--



a) $f = 2$, $m = 3$, $dip = 9$　b) $f = 2$, $m = 5$, $dip = 7$　c) $f = 2$, $m = 7$, $dip = 5$

d) $f = 3$, $m = 3$, $dip = 14$　e) $f = 3$, $m = 5$, $dip = 8$　f) $f = 3$, $m = 7$, $dip = 5$

g) $f = 5$, $m = 3$, $dip = 13$　h) $f = 5$, $m = 5$, $dip = 7$　i) $f = 5$, $m = 7$, $dip = 5$

图 3.3　DLPV-FM 启发式与问题下界的比较结果

Fig. 3.3　Computational results between DLPV-FM and LB

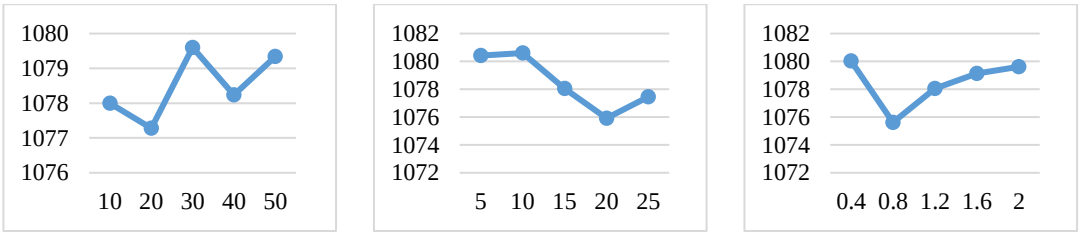
3.5.3　离散人工蜂群算法数值实验

为了使比较结果更具说服力，本研究使用 Ruiz 在 http://soa.iti.es/rruiz 上发表的分布式置换流水问题的基准集进行数值实验。将会测试处理单元数、任务数和处理机数分别为 $f = \{2, 3, 4, 5, 6, 7\}$，$n = \{20, 50, 100, 200, 500\}$，$m = \{5, 10, 20\}$的 50 种不同组合。由于 Naderi 和 Ruiz[10]提出的 $DF_m \mid prmu \mid C_{max}$ 问题与 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题存在差异，在生成数据时为了适应所求问题，增加了释放时间和处理单元速度。为了模拟实际情况，

在生成任务的释放时间按照一定的到达比例，其中释放后可以直接处理的任务与需要等待的任务的比例保持在 1/3 到 1。对于其中一半的处理单元，处理速度简单地设置为 2，而对于其他处理单元，处理速度设置为 1。

(1) 参数设置

为了释放 DABC 算法的完整性能，对种群规模 *Pop*，放弃阈值 AC，调整参数 β 这三个参数进行了正交试验，用实验计划法(Design of Experiments, DOE)，在处理单元数为 4，处理机数为 5，任务数为 100 的规模下进行了正交试验，取值范围分别为：*Pop* = {10, 20, 30, 40, 50}，*AC* = {5, 10, 15, 20, 25}，β = {0.4, 0.8, 1.2, 1.6, 2}，其结果如图 3.4 所示。因此，参数定为 *Pop* = 20，*AC* = 20，β = 0.8。



a) *Pop* 实验结果　　b) *AC* 实验结果　　c) β 实验结果

图 3.4　*Pop*, *AC* 和 β 均值-主效应图

Fig. 3.4　Main Effects Plot of *Pop*, *AC* and β

(2) DABC 算法组件分析

本小节将测试三种改进策略在 DABC 算法中的影响。将改进后的策略分别替换为经典策略进行了验证。这些对比算法包括：

$DABC_1$：无 RA 规则的 DABC 算法；

$DABC_2$：无接受劣解策略的 DABC 算法。

测试结果如表 3.3 所示，每个表上的数值都是该算例运行五次的结果所取的均值，表上的案例按照公式 $m \cdot n / f$ 的结果从小到大排序，可以看到 $DABC_1$ 对算法的改进率是随规模递增的（在 $n = 50$ 时的 0.32%到 $n = 500$ 时的 11.40%），$DABC_2$ 偏向于在规模较小的情况下有着较好的表现（在 $n = 50$ 时的 8.34%到 $n = 500$ 时的 0.94%）。这表明，在规模较小时，有概率接受劣解的策略能使得算法更不容易在邻域搜索时陷入局部最优；在规模较大时，RA 规则使算法在搜索的时候避免作业进入未释放的区域，使算法可以有方向地寻优，减少了过程中的算力浪费。本文提出的策略在不同规模形成互补，使算法在各个规模都能有稳定的发挥。

表 3.3　DABC 算法组件分析结果

Tab. 3.3　Results of the test strategies for DABC

规模 f - n - m	$m \cdot n / f$	DABC	$DABC_1$		$DABC_2$	
			Value	Gap	Value	Gap
6-50-5	41.67	494.9	498.5	0.73%	528.4	6.77%
6-50-10	83.33	692.9	695.1	0.32%	750.7	8.34%
6-100-10	166.67	1014.2	1034.2	1.97%	1076.7	6.16%
5-100-10	200.00	1209.4	1221.7	1.02%	1285.6	6.30%
2-100-5	250.00	1850.8	1876.8	1.40%	1912	3.31%
7-200-10	285.71	1593.4	1617.1	1.49%	1683.6	5.66%
3-100-10	333.33	1933.2	1963.8	1.58%	2008.2	3.88%
3-200-20	1333.33	4000	4070	1.75%	4110.8	2.77%
2-200-20	2000.00	5138.8	5337.2	3.86%	5175	0.70%
4-500-20	2500.00	6295.5	7012.9	11.40%	6354.8	0.94%

*Gap = (Value($DABC_x$) - Value(DABC)) / Value(DABC)

(3) 与其他智能优化算法对比

本小节将用所提出的 DABC 算法与 sIG 算法[22]，HABC 算法[25]和 EGA 算法[32]进行实验对比。为保证实验的公平，所有算法都使用前节 3.4.2 所提出的初始化策略进行初始化。

对于每一组算例，每个算法都运行 5 次并取均值作为最终结果，每次运行 $f \times m \times n \times 50$（毫秒）CPU 时间。采用 RDP(Relative Difference Percentage)指标来评价结果，$RDP = (Z^{ALG} - Z^{BEST})/Z^{BEST} \times 100\%$，其中，$Z^{ALG}$ 表示上述四个算法运行出来的结果，Z^{BEST} 表示这四个算法中最好的那个结果。

实验结果如表 3.4 所示。可以明显看出，在绝大部分规模下，本文提出的 DABC 算法都获得了最好的结果。在 50 组算例中，DABC 算法只在其中两个算例中没有比过 sIG 算法与 HABC 算法，并且最大 RDP 值只有 0.38%。

表 3.4　DABC、sIG、HABC 和 EGA 的对比结果

Tab. 3.4　The comparison results for DABC, sIG, HABC and EGA

scale	DABC		sIG		HABC		EGA		CPUtime/s
f - n - m	Value	RDP	Value	RDP	Value	RDP	Value	RDP	
2-20-5	**501**	0.00%	504.3	0.66%	501.6	0.12%	536.5	7.09%	10
2-20-10	741.2	0.30%	**739**	0.00%	740.1	0.15%	772.6	4.55%	20
2-50-10	**1279.5**	0.00%	1327.4	3.74%	1318	3.01%	1417.8	10.81%	50
2-50-20	**1779.7**	0.00%	1818.4	2.17%	1818.1	2.16%	1942.2	9.13%	100
2-100-5	**1850.8**	0.00%	1985.3	7.27%	1953.9	5.57%	2063.6	11.50%	50

表 3.4　DABC、sIG、HABC 和 EGA 的对比结果（续）

scale	DABC		sIG		HABC		EGA		CPUtime/s
f - *n* - *m*	Value	RDP	Value	RDP	Value	RDP	Value	RDP	
2-100-10	**2267.9**	0.00%	2373.8	4.67%	2365	4.28%	2489.4	9.77%	100
2-200-10	**4016.8**	0.00%	4268.2	6.26%	4211.3	4.84%	4428.8	10.26%	200
2-200-20	**5138.8**	0.00%	5642.3	9.80%	5608.3	9.14%	5414.8	5.37%	400
3-20-5	445.1	0.38%	448.3	1.11%	**443.4**	0.00%	487.6	9.97%	15
3-20-10	**696**	0.00%	702.4	0.92%	696.2	0.03%	752.7	8.15%	30
3-20-20	**1014**	0.00%	1015	0.10%	1014.4	0.04%	1044	2.96%	60
3-50-5	**905.2**	0.00%	931.1	2.86%	920.8	1.72%	1026.5	13.40%	37.5
3-50-10	**1162**	0.00%	1201.1	3.36%	1201.5	3.40%	1331.4	14.58%	75
3-100-10	**1933.2**	0.00%	2054.5	6.27%	2073.6	7.26%	2239.6	15.85%	150
3-100-20	**2442.8**	0.00%	2532.6	3.68%	2604.9	6.64%	2825.2	15.65%	300
3-200-10	**3258.4**	0.00%	3482.9	6.89%	3482.7	6.88%	3714.3	13.99%	300
3-200-20	**4000**	0.00%	4332.1	8.30%	4474.3	11.86%	4682	17.05%	600
4-20-10	**539.1**	0.00%	545.5	1.19%	546.9	1.45%	583.1	8.16%	40
4-50-20	**1194.6**	0.00%	1240	3.80%	1230.7	3.02%	1413.1	18.29%	200
4-100-5	**1077.5**	0.00%	1137	5.52%	1130.2	4.89%	1242.3	15.29%	100
4-200-10	**2291.1**	0.00%	2472.2	7.90%	2449.5	6.91%	2618.2	14.28%	400
4-200-20	**3099.9**	0.00%	3297.5	6.37%	3379.7	9.03%	3676.3	18.59%	800
4-500-20	**6295.5**	0.00%	7220.7	14.70%	7055.3	12.07%	7322.8	16.32%	2000
5-20-10	**506.6**	0.00%	512	1.07%	511.6	0.99%	561.2	10.78%	50
5-20-20	**832.4**	0.00%	834.8	0.29%	836.2	0.46%	888.8	6.78%	100
5-50-5	**539.7**	0.00%	563.2	4.35%	561.9	4.11%	653	20.99%	62.5
5-50-20	**1248**	0.00%	1298.2	4.02%	1279.8	2.55%	1500.1	20.20%	250
5-100-5	**1007.9**	0.00%	1073.7	6.53%	1065	5.67%	1182.3	17.30%	125
5-100-10	**1209.4**	0.00%	1302.5	7.70%	1303.9	7.81%	1500.1	24.04%	250
5-100-20	**1786.5**	0.00%	1874.1	4.90%	1969.6	10.25%	2254.9	26.22%	500
5-200-20	**2723.4**	0.00%	2870.4	5.40%	3031.5	11.31%	3352.7	23.11%	1000
6-20-5	**241.6**	0.00%	246.4	1.99%	253	4.72%	291.6	20.70%	30
6-20-10	**430.6**	0.00%	434.8	0.98%	444.4	3.20%	482.5	12.05%	60
6-20-20	**755.5**	0.00%	764.3	1.16%	758.9	0.45%	856.3	13.34%	120
6-50-5	**494.9**	0.00%	522.1	5.50%	522.4	5.56%	596.5	20.53%	75
6-50-10	**692.9**	0.00%	732	5.64%	741.4	7.00%	850.9	22.80%	150
6-50-20	**999.7**	0.00%	1032.5	3.28%	1027.4	2.77%	1248.3	24.87%	300
6-100-5	**802.9**	0.00%	854.3	6.40%	846.9	5.48%	938.8	16.93%	150
6-100-10	**1014.2**	0.00%	1100.3	8.49%	1148.7	13.26%	1289.5	27.14%	300
6-100-20	**1511**	0.00%	1591.1	5.30%	1610.2	6.57%	1975.6	30.75%	600
6-200-10	**1731**	0.00%	1829.7	5.70%	1865.4	7.76%	2066.9	19.40%	600
7-20-5	**269.2**	0.00%	272.5	1.23%	278.7	3.53%	325.1	20.77%	35
7-20-10	**447.3**	0.00%	452	1.05%	450.3	0.67%	513.5	14.80%	70

表 3.4　DABC、sIG、HABC 和 EGA 的对比结果（续）

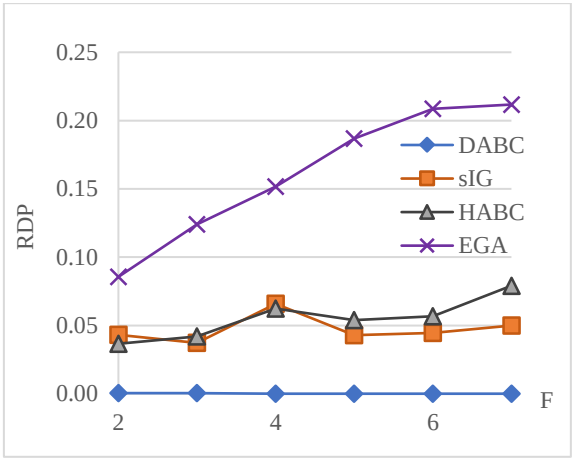
scale	DABC		sIG		HABC		EGA		CPUtime/s
f - *n* - *m*	Value	RDP	Value	RDP	Value	RDP	Value	RDP	
7-20-20	**700.5**	0.00%	717.1	2.37%	707.6	1.01%	803.7	14.73%	140
7-50-10	**649.9**	0.00%	683.2	5.12%	692.8	6.60%	838.8	29.07%	175
7-50-20	**999.2**	0.00%	1041.6	4.24%	1049.9	5.07%	1253.3	25.43%	350
7-100-5	**785.8**	0.00%	827.9	5.36%	848.5	7.98%	929	18.22%	175
7-200-10	**1593.4**	0.00%	1735.1	8.89%	1791.8	12.45%	1943.1	21.95%	700
7-200-20	**2135.3**	0.00%	2255.6	5.63%	2583.3	20.98%	2732.9	27.99%	1400
7-500-20	**4239.1**	0.00%	4706.9	11.04%	4784.3	12.86%	4985.1	17.60%	3500

表 3.5　根据 *f*、*n* 和 *m* 划分的平均 RDP

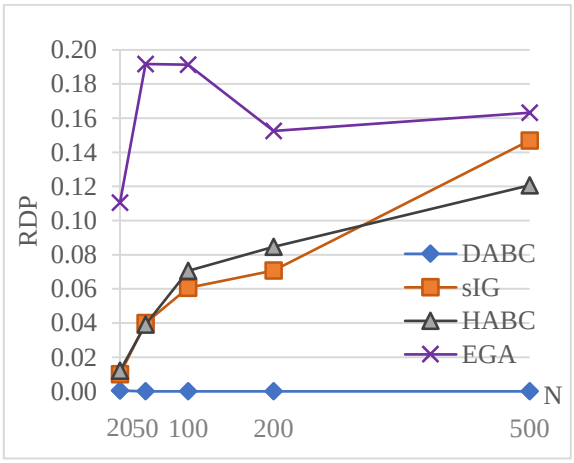
Tab. 3.5　Average RDP of the Computational results according to *f*, *n* and *m*

		DABC	sIG	HABC	EGA
f	2	0.0004	0.0432	0.0366	0.0856
	3	0.0004	0.0372	0.0420	0.1240
	4	0	0.0658	0.0623	0.1516
	5	0	0.0428	0.0539	0.1868
	6	0	0.0444	0.0568	0.2085
	7	0	0.0499	0.0791	0.2117
m	5	0.0003	0.0406	0.0411	0.1606
	10	0.0002	0.0452	0.0516	0.1539
	20	0	0.0508	0.0675	0.1760
n	20	0.0005	0.0101	0.0120	0.1106
	50	0	0.0401	0.0391	0.1917
	100	0	0.0607	0.0706	0.1913
	200	0	0.0708	0.0847	0.1526
	500	0	0.1470	0.1207	0.1632
总平均		0.0001	0.0535	0.0584	0.1620

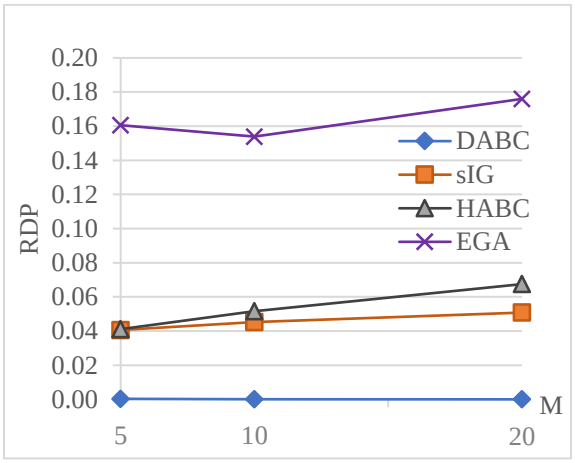
为了进一步分析，对于 RDP 结果分别按 *f*、*n*、*m* 进行平均分组，其数值结果在表 3.5 中。可以看出本文提出的 DABC 算法的平均 RDP 值基本为 0，在每个规模中都处于平均 RDP 值最低的状态。图 3.5 将结果呈现得更加直观，可以看出，EGA 算法的性能较差，在受到工厂数量 *f* 的影响更大（$f = 2$ 时为 8.56%，$f = 7$ 时为 21.17%）。sIG 算法和 HABC 算法的效率随着任务数量 *n* 规模的增加类似地增长，并逐渐趋于平缓（从 $n = 20$ 时的 1.01%、1.20%到 $n = 500$ 时的 14.70%、12.07%），而在中等规模的情况下，EGA 算法显得相当不稳定。此外，所有涉及的算法基本上不受处理机数量 *m* 的影响。综上所述，本文提出的 DABC 算法能够在相同的 CPU 时间限制下为 $DQ_m \mid r_j, prmu \mid C_{max}$ 问题提供更好的解。



a) 根据 *f* 分组的平均 RDP



b) 根据 *n* 分组的平均 RDP



c) 根据 *m* 分组的平均 RDP

图 3.5　根据 *f*、*n* 和 *m* 分组的平均 RDP 结果图

Fig. 3.5　Average RDP of the Computational results according to *f*, *n* and *m*

3.6　本章小结

本章研究了目标为极小化最大完工时间，考虑到释放时间和处理单元速度的分布式置换流水作业调度问题。建立了混合整数规划模型来描述这一强 NP 难问题。为了获得最优解，提出了一种具有有效上界、下界和剪枝策略的分支定界算法。并利用 CLPEX 求解器进行求解，与分支定界算法进行比较。实验结果表明，所提出的分支定界算法具有较高的性能。设计的 DLPV-FM 启发式能在可接受的时间内解决大规模问题，与下界的比较结果表明，DLPV-FM 启发式算法具有收敛性。同时，使用 DABC 算法对中等规模案例进行求解，设计了随机可用规则和接受策略来提高 DABC 算法的性能。最后，通过已发表的数据集中的案例进行了仿真测试，结果表明了本文所提出算法的优越性。

4 考虑释放时间的分布式双代理开放作业调度问题

4.1 问题介绍与模型构建

分布式开放作业问题模型是分布式服务中最常用的模型，每个任务在被分配至处理单元后，可以以任意顺序经过处理机进行处理。根据代理介入的情况，考虑了任务所属代理的优先级，建立了双代理模型。为了使问题模型更具有现实意义，在模型中引入了释放时间的概念，充分考虑了任务到达时间不一致的情况，使本问题模型更加灵活。

4.1.1 问题描述

在 $DO_m|r_j|\ C_{\max}^{A}+\theta C_{\max}^{B}$ 问题中，包括 f 个处理单元，m 台处理机，n 个任务，其中一部分任务属于 A 代理，另一部分属于 B 代理。每个任务拥有自己的释放时间 r_j 和在每一台处理机上的处理时间 $p_{i,j}$，并且可以去任意一个处理单元，以任意顺序经过 m 台处理机进行作业。每台处理机一次只能处理一个任务，每一个任务同时只能被一台处理机处理。假设处理机之间的存储空间足够，即不会产生阻塞。C_{max}^{A} 与 C_{max}^{B} 分别为代理 A 的最大完工时间与代理 B 的最大完工时间。本问题的优化目标是找到一个序列，使代理 A 任务与代理 B 任务的最大完工时间加权和最小。

4.1.2 混合整数规划模型

本节介绍了一个混合整数规划模型用于描述问题。其中所涉及的其他参数与变量如下所示。

参数

n	总任务数
f	总处理单元数
m	总处理机数
l	总位置数
Y	一个极大的正整数
$p_{i,j}$	任务 j 在处理机 i 上的处理量/时间
r_j	任务 j 的释放时间
n^A	代理 A 的总任务数（在任务集合中前 n^A 个为代理 A 的任务，n^A 后的任务为代理 B 的任务）

集合

N	任务集合，索引为 $j\in\{1,\ldots,n\}$

M　　处理机集合，索引为 $i \in \{1,\ldots,m\}$

F　　处理单元集合，索引为 $h \in \{1,\ldots,f\}$

L　　位置集合，索引为 $k \in \{1,\ldots,l\}$

变量

$y_{j,h}$　　=1，如果任务 j 在处理单元 h 上；=0，否则

$x^{i}_{j_1,j_2}$　　=1，如果处理机 i 上的处理顺序是从任务 j_1 到任务 j_2；=0，否则

x^{i_1,i_2}_{j}　　=1，如果任务 j 处理顺序是从处理机 i_1 到处理机 i_2 上；=0，否则

$S_{i,j}$　　表示任务 j 在处理机 i 上的开工时间

$C_{i,j}$　　表示任务 j 在处理机 i 上的完工时间

本模型基于任务位置构建，通过前后任务的时间窗对任务进行约束，使所得解可行，具体如下所示。

$$Minimize \quad C^{A}_{max} + \theta C^{B}_{max}$$

$$Subject\ to:$$

$$\sum_{h=1}^{f} y_{j,h} = 1,\ j \in N \tag{4.1}$$

$$x^{i_1,i_2}_{j} + x^{i_2,i_1}_{j} = 1, i_1, i_2 \in M,\ j \in N, i_1 \neq i_2 \tag{4.2}$$

$$x^{i}_{j_1,j_2} + x^{i}_{j_2,j_1} = 1, i \in M,\ j_1, j_2 \in N,\ j_1 \neq j_2 \tag{4.3}$$

$$C_{i_2,j} - C_{i_1,j} \geq p_{i_2,j} - Y(2 - x^{i_1,i_2}_{j} - y_{j,h}), i_1, i_2 \in M,\ j \in N, h \in F, i_1 \neq i_2 \tag{4.4}$$

$$C_{i,j_1} - C_{i,j_2} \geq p_{i,j_2} - Y(3 - x^{i}_{j_1,j_2} - y_{j_1,h} - y_{j_2,h}), i \in M,_1,\ j_2 \in N, h \in F,\ j_1 \neq j_2 \tag{4.5}$$

$$r_j + p_{i,j} \cdot y_{j,h} \leq C_{i,j}, i \in M,\ j \in N, h \in F \tag{4.6}$$

$$C^{A}_{max} \geq C_{i,j}, i \in M,\ j \in \{1..n^{A}\} \tag{4.7}$$

$$C^{B}_{max} \geq C_{i,j}, i \in M,\ j \in \{n^{A}+1..n\} \tag{4.8}$$

$$y_{j,h}, x^{i}_{j_1,j_2}, x^{i_1,i_2}_{j} \in \{0,1\}; C_{i,j} \geq 0; r_j \geq 0; p_{i,j} \geq 0 \tag{4.9}$$

约束(4.1)确保每个任务只能在一个处理单元处理；约束(4.2)限制了同一个任务在不同处理机之间经过的顺序；约束(4.3)限制了不同任务在同一处理机上的作业顺序；约束(4.4)表示同一任务在相邻处理机上完工时间关系；约束(4.5)表示同一处理机上相邻任务的完工时间关系；约束(4.6)定义了任务释放时间与完工时间的关系；约束(4.7) (4.8)定义了两个代理的最大完成时间；约束(4.9)确定了变量和参数的范围。

4.2 EAF-ADA-DS 启发式算法设计与流程

对于 $DO_m \mid r_j \mid C^A_{\max} + \theta C^B_{\max}$ 问题，考虑到双代理的特性，使用优势代理优先原则会有较好的效果，在计算的过程中，使 A 代理与 B 代理分别作为优势代理进行计算，选择最终结果较好的作为最终优势代理。

在算法运行过程中，首先使用 EAF(Early Available Factory)规则，选择当前可用的最早处理单元。再使用当前可用优势代理优先(Available Dominant Agency, ADA)规则，对于当前可用的任务，优先选择优势代理的任务进行处理。DS 算法表示当前有任务释放便直接进行处理，这有时会与 ADA 冲突，存在当前到来非优势代理任务，但在该任务处理结束之前，便有优势代理任务到来的情况发生。在经过数值实验对比后，选择了以 ADA 为主，DS 为辅的启发式 ADA-DS 算法，并且在当前有多个同代理任务释放的时候使用 SPT 规则进行选择[72]。其过程如伪代码 4.1 所示。

伪代码 4.1：**EAF-ADA-DS 启发式**

```
1  Begin
2  初始化当前优势代理 DA
3  For DA = 1 to 2 do
4      For d = 1 to m*n do
5          初始化当前处理单元索引 cur_f
6          将 cur_f 更新为当前任务数最少的处理单元（任务数相等则选择工序数最少的处理单元）
7          初始化当前处理机索引 cur_m
8          初始化当前任务索引 cur_j
9          IF 当前有可用工序且已在处理单元 cur_f 或还没分配处理单元
10             更新当前任务为 cur_j
11             IF 任务 cur_j 有工序未排序
12                 更新该工序所在阶段为 cur_m
13             Endif
14             对于已释放的当前工序，与其他所有工序进行对比，假如当前工序为非优势代理工序，存在其他已释放的优势代理工序，则更新当前工序；假如两个工序为同个代理，且其他工序已作业的工序多，则更新当前工序；假如工序数相等，且其他工序总处理时间较短，则更新当前工序；假如总处理时间相等，且其他工序当前阶段处理时间较短，则更新当前工序
15         Elseif 当前没有可用工序
16             找到释放时间最早的作业为当前工序，释放时间相同情况下，假如当前工序为非优势代理，其他工序为优势代理，则更新当前工序；假如是同代理，且其他工序已处理工序数量多，则更新当前工序；假如工序数相等，且其他工序总处理时间较短，则更新当前工序；假如总处理时间相等，且其他工序当前阶段处理时间较短，则更新当前工序
```

伪代码 4.1：**EAF-ADA-DS 启发式**

```
            Endif
            更新当前工序为已用
            计算当前处理机时间
        Endfor
    Endfor
    计算目标函数
    End
```

4.3 下界设计与流程

在求解下界的过程中，将原问题中，任务一旦开始处理便不可打断的约束松弛掉，分别以 A 代理与 B 代理作为优势代理进行下界计算，运用可中断的 ADA-DS 算法对每个阶段进行单机排序，比较其该阶段结果大小，结果较小的代理作为该阶段的优势代理。并将以该代理为优势代理的结果储存为该阶段的下界值，最后选出各个阶段中较大的下界值作为最终下界值。过程如伪代码 4.2 所示，其中 *DAT* 为优势代理作业完工时间，*NDAT* 为非优势代理作业完工时间。

伪代码 4.2： **DO_m 问题下界**

```
Input: 阶段数 m，优势代理 DA
Begin
T←0
DAT←0
NDAT←0
将任务按照释放时间从小到大排序
For j = 1 to n do
If j 的代理为 DA
    If T ≥ r_j
        T + remain_p_{m,j} ←T
        更新任务 j 的剩余加工时间为 0
    Else
        For k = 0 to j do
            If T 加任务 k 的剩余处理时间小于下个优势代理的释放时间
                更新 T：加上任务 k 的剩余加工时间
                更新任务 k 的剩余加工时间为 0
            Else
                更新 T 为下一个优势代理的释放时间
```

伪代码 4.2： **DO_m 问题下界**

```
                    更新任务 k 的剩余加工时间
                Endif
            Endfor
        Endif
        DAT ← T
Else
        获得下一个优势代理任务的释放时间 NextR
        If T ≥ NextR
            Continue
        Else
        If T ≥ r_j
            If T + remain_p_mj ≤ NextR
                更新 T：加上任务 k 的剩余处理时间
                更新任务 k 的剩余处理时间为 0
            Else
                更新 T 为下一个优势代理的释放时间
                更新作业 k 的剩余处理时间
            Endif
        Else
            For k = 0 to j do
                If T 加任务 k 的剩余处理时间小于下个优势代理的释放时间
                    更新 T：加上任务 k 的剩余处理时间
                    更新任务 k 的剩余处理时间为 0
                Else
                    更新 T 为下一个优势代理的释放时间
                    更新任务 k 的剩余处理时间
                Endif
            Endfor
            If T + remain_p_mj ≤ NextR
                更新 T：加上任务 k 的剩余处理时间
                更新任务 k 的剩余处理时间为 0
            Else
                更新 T 为下一个优势代理的释放时间
                更新任务 k 的剩余处理时间
            Endif
        Endif
```

伪代码 4.2： **DO_m 问题下界**
55 $NDAT \leftarrow T$
56 **Endif**
57 **Endfor**
58 在最后补上前面没有放进去的非优势代理任务的剩余处理时间
59 根据目标函数计算 LB
60 **End**

4.4 离散差分进化算法

DE 算法一般用于处理连续型问题，而 $DO_m|r_j|\ C_{\max}^{A}+\theta C_{\max}^{B}$ 问题为离散型问题，因此本文将使用离散差分进化(Discrete Differential Evolution, DDE)算法来计算。

4.4.1 编码与解码

对于 $DO_m|r_j|\ C_{\max}^{A}+\theta C_{\max}^{B}$ 问题，需要考虑两个子调度：一是各处理单元的任务分配；二是各处理单元内工序的调度顺序。在调度过程中，由于各个工序可以按照不同的顺序经过处理机，本问题的编码在处理机与任务的维度上就已经需要用到二维数组，若将工处理单元维度加入编码则有些累赘。因此，在编码过程中，将不考虑处理单元维度，使用二维矩阵对处理机与任务进行编码，行表示各个处理机，列表示任务即将被调度的优先级。

与流水作业编码不同，开放作业的任务编码，是在解码过程中，作业将被抽取的优先级，而非任务在处理机上即将处理的顺序。编码没有直接考虑处理单元，这就需要在解码过程中考虑处理单元的任务分配问题，因此在开始解码时，从第一个处理机开始解码，会先根据处理单元的当前负载量分配任务，任务会被优先分配在已有任务数较少的处理单元中，之后就按照编码顺序计算每个任务的完工时间，计算目标函数。

4.4.2 初始化策略

DDE 算法是基于种群的智能优化算法，初始种群的生成对于后续的寻优起着重要的作用。对于一个开放作业问题，最优的调度片段有极大的可能存在于一个劣解中，因此对于 $DO_m|r_j|\ C_{\max}^{A}+\theta C_{\max}^{B}$ 问题的初始化种群，$Pop/2$ 的种群解会根据 RA 规则生成（Pop 为种群规模），剩余部分由前节 4.2 提到的 EAF-ADA-DS 启发式算法解与完全随机的初始解来组成。

4.4.3 变异操作

在变异阶段，会生成一个用来扰动目标个体的变异个体。变异算子会作用于当前种群的每一个个体，并生成变异个体。进行变异操作时，会选择四个不同的个体进行矢量差分，如公式(4.10)所示。

$$V_i^g = X_{best}^{g-1} \oplus vr \otimes \left(X_{a_1}^{g-1} - X_{b_1}^{g-1} \right) \oplus vr \otimes \left(X_{a_2}^{g-1} - X_{b_2}^{g-1} \right) \tag{4.10}$$

其中，X_{best}^{g-1} 表示第 $g-1$ 代种群中的最优个体，$X_{a_1}^{g-1}$，$X_{b_1}^{g-1}$，$X_{a_2}^{g-1}$，$X_{b_2}^{g-1}$ 表示第 $g-1$ 代种群中的四个随机个体，在公式(4.10)中的运算符⊗的运算方法如公式(4.11)所示。

$$G_{i_x}^g = vr \otimes \left(X_{a_x}^{g-1} - X_{b_x}^{g-1} \right) = \begin{cases} X_{a_x}^{g-1} - X_{b_x}^{g-1}, \mu < vr \\ 0 \qquad\qquad ,\text{否则} \end{cases} \tag{4.11}$$

其中，$X_{a_x}^{g-1}$ 和 $X_{b_x}^{g-1}$ 为第 $g-1$ 代种群中的两个不同个体，$x = \{1, 2\}$。通过比较变异概率 vr 与随机概率 $\mu = \text{rand}(0, 1)$可得加权差分个体 $G_{i_x}^g$。如果 μ 小于等于变异概率 vr 则保留该编码，否则置对应的编码为零。公式(4.10)中的运算符⊕的运算方法如公式(4.12)所示。

$$V_i^g = X_{best}^{g-1} \oplus G_{i_1}^g \oplus G_{i_2}^g = \text{mod}\left(\left(X_{best}^{g-1} + G_{i_1}^g + G_{i_1}^g \right), n \right) + 1 \tag{4.12}$$

其中，V_i^g 为最终的变异个体，mod()是进行取模运算。进行差分后的个体可能会存在编码为零、负值或者大于取值范围的情况，因此需要取模来进行修补，将编码映射到合理范围内。

4.4.4 交叉操作

为了增加个体的扰动，本文采用三点交叉的交叉操作。变异个体 V_i^g 会与目标个体 X_i^{g-1} 生成一个测试个体 U_i^g。在交叉过程中，首先也会如变异操作一样为每个编码生成一个随机的交叉概率 $\gamma = \text{rand}(0, 1)$，若 γ 大于交叉概率 cr，则对应编码不参与交叉，赋值为–1；否则，保留该编码。在目标个体 X_i^{g-1} 中随机选择三个插入位置，将变异个体 V_i^g 分为三部分进行插入，得到测试个体 U_i^g。

由于交叉后的测试个体 U_i^g 中可能存在重复编码，因此，需要对其进行去重操作，即删去重复的编码，对于重复的编码，都保留首次出现的，删去之后出现的，得到最终的个体 X_i^g。

4.4.5 离散差分进化算法流程

在 DDE 算法的选择阶段，将进行贪婪的选择，假如最终个体 X_i^g 的适应度值比目标个体 X_i^{g-1} 好，那么个体 X_i^g 将替代个体 X_i^{g-1}，否则，保留原来的个体 X_i^{g-1}。整个离散差分进化算法过程如伪代码 4.3 所示。

伪代码 4.3：**离散差分进化算法**

```
1  Begin
2  初始化种群
3  tatalT ← m·n·f·50
4  curT ← 0
5  While curT ≤ totalT do
6      变异操作( X_{a1}^{g-1}, X_{b1}^{g-1}, X_{a2}^{g-1}, X_{b2}^{g-1}, X_{best}^{g-1}, vr) → V_i^g
7      交叉操作( X_i^{g-1}, V_i^g, cr) → U_i^g
8      If f(U_i^g) < f(X_i^{g-1})
9          X_i^g ← U_i^g
10     Else
11         X_i^g ← X_i^{g-1}
12     Endif
13     更新最优个体 X_best
14 Endwhile
15 End
```

4.5 数值仿真实验

为了测试本文所提出的各个算法，进行了严谨的数值实验，下界、EAF-ADA-DS 算法和 DDE 算法都使用 Visual Studio 软件用 C++进行编写，运行于 Intel(R) Core(TM) i5-9400 CPU 2.90GHz，内存 8GB 的 Windows10 操作系统上。

4.5.1 EAF-ADA-DS 启发式算法数值实验

为了验证 EAF-ADA-DS 启发式算法作为大规模情况下启发式算法的性能，本小节设计了详尽的数值实验如下：测试了 f= {2,4,6}，m ={3,5,7}的 9 种不同组合，每个组合随机生成 10 组案例，分别在 n ={30, 50, 100, 200, 500, 1000, 1500, 2000}的规模下运行。

为了更贴近实际，在释放后可以直接处理的作业和需要等待的作业的比例为 1/3 到 1。生成的释放时间均匀分布在[0, $dip \cdot n / m \cdot f$]区间内，其中 dip = [1, 35]根据规模确定。

图 4.1 所示为 $DO_m \mid r_j \mid \ C^A_{\max} + \theta C^B_{\max}$ 问题启发式算法与下界的 GAP 值，其中 GAP = 100%*($C_{\text{EAF-ADA-DS}} - C_{\text{LB}}$) / C_{LB}，可以看出随着规模的扩大，GAP 值逐渐变小，算法具有一定收敛性。

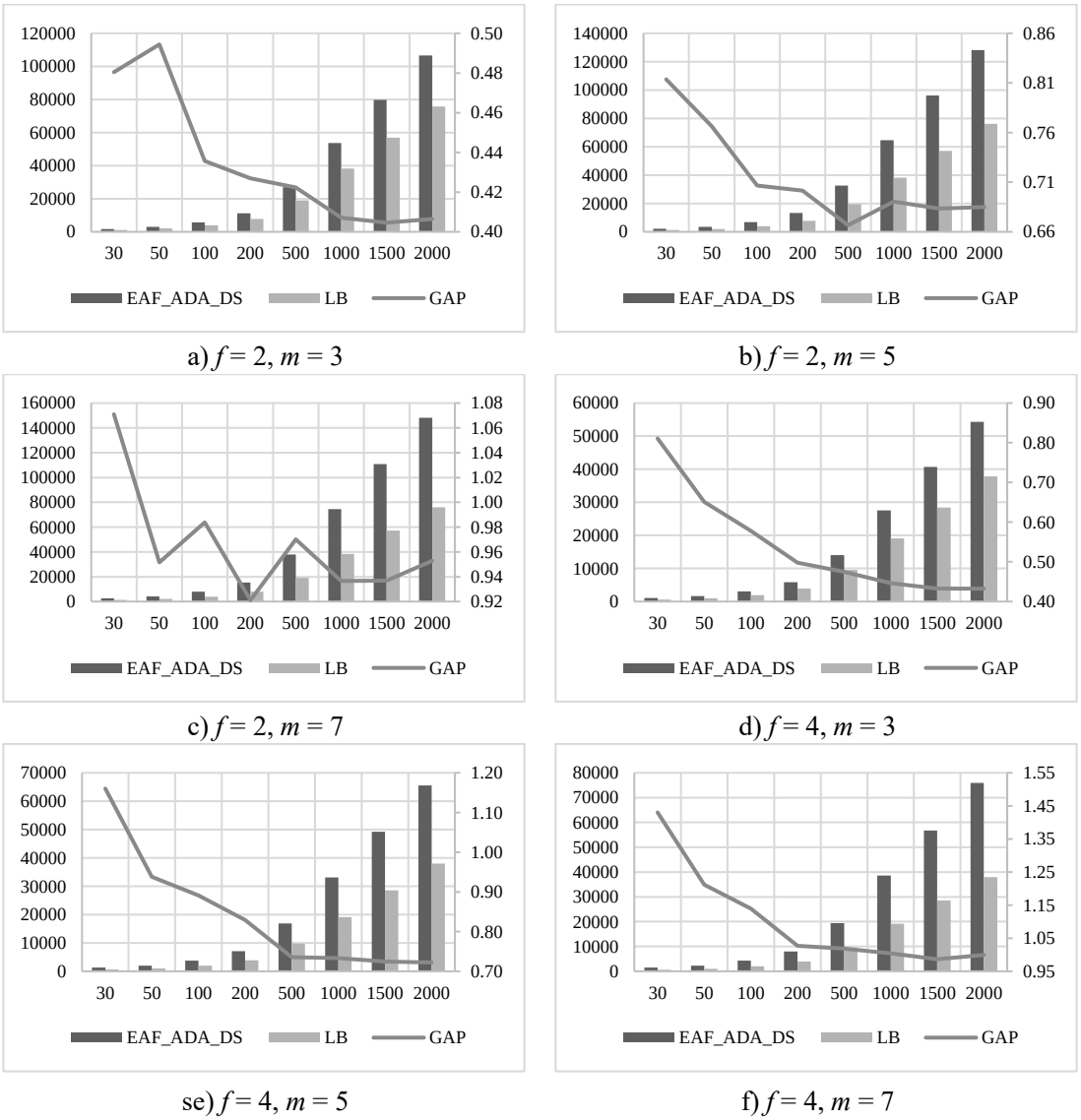
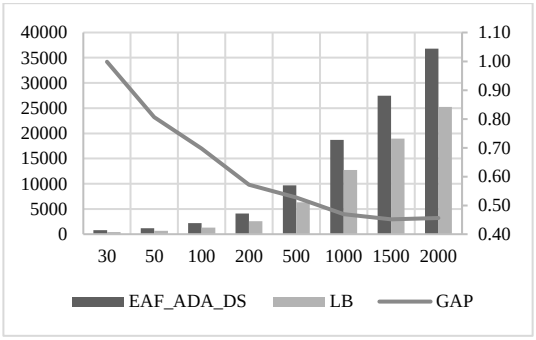
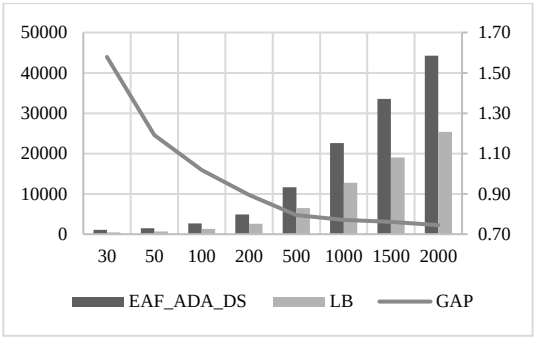


图 4.1　EAF-ADA-DS 与下界对比结果

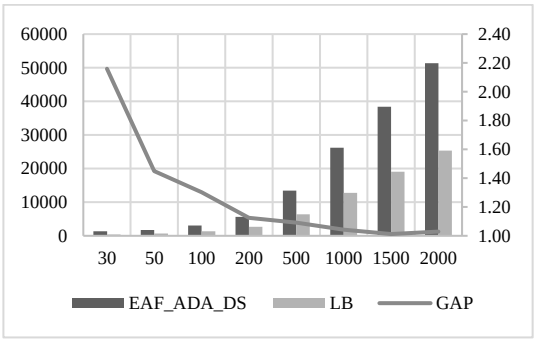
Fig. 4.1　Computational results between EAF-ADA-DS and LB



g) $f = 6, m = 3$



h) $f = 6, m = 5$



i) $f = 6, m = 7$

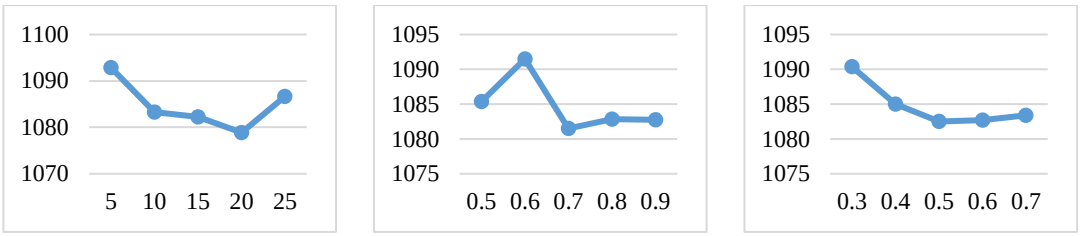
图 4.1　EAF-ADA-DS 与下界对比结果（续）

4.5.2　离散差分进化算法数值实验

为了验证离散差分进化算法的性能，本小节采用数值仿真实验对其进行测试。对于处理单元数、任务数和处理机数分别为 f = {2,4,6}，m ={3,5,7}，n ={10, 20, 30, 50, 100, 200}的 54 种不同组合，每个组合随机生成 10 个案例，每个案例都会运行五次取平均值作为案例结果，每组规模的最终结果为 10 个案例结果的平均值。为了模拟实际情况，在生成任务的释放时间按照一定的到达比例，其中释放后可以直接处理的工序与需要等待的工序的比例保持在 1/3 到 1。

(1) 参数设置

为了释放差分进化算法的完整性能，对种群规模 Pop，变异概率系数 vr，交叉概率系数 cr 这三个参数进行正交试验，用 DOE 法进行田口实验分析，在处理单元数为 2，处理机数为 3，任务数为 10 的规模下，对十组算例进行了正交试验，参数的取值范围分别为：Pop = {5, 10, 15, 20, 25}，vr = {0.5, 0.6, 0.7, 0.8, 0.9}，cr = {0.3, 0.4, 0.5, 0.6, 0.7}，其结果如图 4.2 所示。因此，参数定为 Pop = 20，vr = 0.7，cr = 0.5。



a) *Pop* 实验结果　　b) *vr* 实验结果　　c) *cr* 实验结果

图 4.2　*Pop*, *vr* 和 *cr* 均值-主效应图

Fig. 4.2　Main Effects Plot of *Pop*, *vr* and *cr*

(2) 与其他智能优化算法对比

本小节对所提出的 DDE 算法与其他文献提出的 EGA_OS 算法[47]进行实验对比。测试了 f = {2,4,6}，m ={3,5,7}的 9 种不同组合，每个组合随机生成 10 组案例，分别在 n ={20, 30, 50, 100, 200}的规模下运行。为了更贴近实际，在释放后可以直接处理的作业和需要等待的作业的比例为 1/3 到 1。生成的释放时间均匀分布在[0, $dip \cdot n / m \cdot f$]区间内，其中 dip = [1, 35]根据规模确定。

对于每一组算例，每个算法都运行 5 次并取均值作为最终结果，每次运行 $f \times m \times n \times 50$（毫秒）CPU 时间。采用 RDP(Relative Difference Percentage)指标来评价结果，RDP = $(Z^{ALG} - Z^{BEST})/Z^{BEST} \times 100\%$，其中，$Z^{ALG}$ 表示算法运行出来的结果，Z^{BEST} 表示这两个算法中较好的那个结果。

实验结果如表 4.1 所示。可以明显看出，在绝大部分规模下，本文提出的 DDE 算法都获得了最好的结果。在 54 组数据中，DDE 算法仅有 4 组没有比过 EGA_OS 算法，并且最大 RDP 值只有 6.02%。图 4.3 可以更清晰地展示算法结果，其中 GAP = $(Z^{EGA\_OS} - Z^{DDE}) / Z^{DDE} \times 100\%$，$Z^{EGA\_OS}$ 为 EGA_OS 算法求得该规模 10 组算例的平均值，Z^{DDE} 则为 DDE 算法所得。可以看到随着规模的增大，GAP 值也逐渐增加，DDE 算法在各个规模下表现得更加稳定，呈现出更优越的效果。

为了更精准地判断算法之间的差异以及验证算法的稳定性，本文对上述实验所得 RDP 值进行了方差分析，其方差如表 4.2 所示，分析计算得 F 值为 61.14，P 值为 4.15E-12，P 值远小于 0.05，可见两个算法的性能存在显著差异，DDE 算法性能显著优于 EGA_OS 算法。又知 DDE 算法 RDP 值的方差远小于 EGA_OS 算法，可见 DDE 算法性能稳定，在不同规模算例中都有不俗表现。

表 4.1　DDE 与 EGA_OS 算法的数值实验结果

Tab. 4.1　Numerical experimental results between DDE and EGA_OS

f - *m*	*n*	Value		RDP	
		DDE	EGA_OS	DDE	EGA_OS
2-3	10	**469.68**	536.545	0.00%	14.24%
	20	**1075.9**	1136.3	0.00%	5.61%
	30	**1723.7**	1836.12	0.00%	6.52%
	50	**2939.8**	3325.63	0.00%	13.12%
	100	**5615.8**	7342.42	0.00%	30.75%
	200	**11047.66**	16343.81	0.00%	47.94%
2-5	10	**650.24**	796.5	0.00%	22.49%
	20	**1413.66**	1729.92	0.00%	22.37%
	30	**2272.5**	2900.02	0.00%	27.61%
	50	**3585.4**	5266.96	0.00%	46.90%
	100	**6805.4**	11470.62	0.00%	68.55%
	200	**13203.66**	25060.08	0.00%	89.80%
2-7	10	**828.5**	1042.78	0.00%	25.86%
	20	**1758.2**	2398.14	0.00%	36.40%
	30	**2584**	3910.3	0.00%	51.33%
	50	**4062.1**	6974.08	0.00%	71.69%
	100	**7982.8**	15547.38	0.00%	94.76%
	200	**15265**	33985.06	0.00%	122.63%
4-3	10	**271.92**	289.78	0.00%	6.57%
	20	**548.84**	580.58	0.00%	5.78%
	30	925.9	**876.2**	5.67%	0.00%
	50	**1621.7**	1639.2	0.00%	1.08%
	100	**3084.5**	3575.02	0.00%	15.90%
	200	**5847.9**	7889.6	0.00%	34.91%
4-5	10	**383.42**	400.28	0.00%	4.40%
	20	**718.02**	842.38	0.00%	17.32%
	30	**1350.8**	1490.78	0.00%	10.36%
	50	**1966.5**	2637.66	0.00%	34.13%
	100	**3770.6**	5691.34	0.00%	50.94%
	200	**7135**	12306.38	0.00%	72.48%
4-7	10	**491.46**	550.18	0.00%	11.95%
	20	**922.66**	1173.2	0.00%	27.15%
	30	**1516.1**	1986.38	0.00%	31.02%
	50	**2301.9**	3495.42	0.00%	51.85%
	100	**4302.9**	7734.82	0.00%	79.76%
	200	**8054.4**	16706.44	0.00%	107.42%

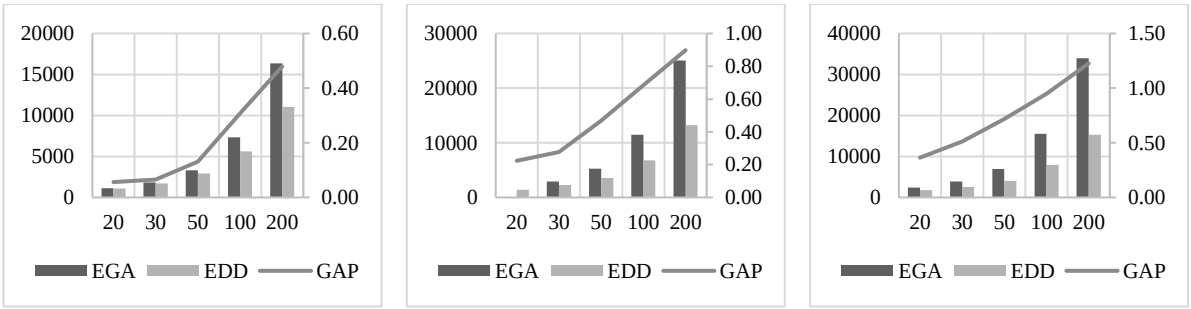
表 4.1　DDE 与 EGA_OS 算法的数值实验结果（续）

f - *m*	*n*	Value		RDP	
		DDE	EGA_OS	DDE	EGA_OS
6-3	10	223.06	**220.57**	1.13%	0.00%
	20	**355.74**	385.67	0.00%	8.41%
	30	**530.38**	589.34	0.00%	11.12%
	50	1142.72	**1077.84**	6.02%	0.00%
	100	**2211.3**	2352.5	0.00%	6.39%
	200	**4090.3**	5290.32	0.00%	29.34%
6-5	10	**268.32**	270.64	0.00%	0.86%
	20	584.24	**569.32**	2.62%	0.00%
	30	**807.72**	943.08	0.00%	16.76%
	50	**1482.1**	1743.88	0.00%	17.66%
	100	**2681.8**	3837.2	0.00%	43.08%
	200	**4925.4**	8404.68	0.00%	70.64%
6-7	10	**335.38**	336.42	0.00%	0.31%
	20	**714.04**	822.8	0.00%	15.23%
	30	**994.76**	1299.92	0.00%	30.68%
	50	**1699.3**	2337.96	0.00%	37.58%
	100	**3088.7**	5142.32	0.00%	66.49%
	200	**5627.9**	11173.86	0.00%	98.54%

表 4.2　DDE 与 EGA_OS 算法 RDP 值方差结果

Tab. 4.2　The variance of the RDP value of DDE and EGA_OS

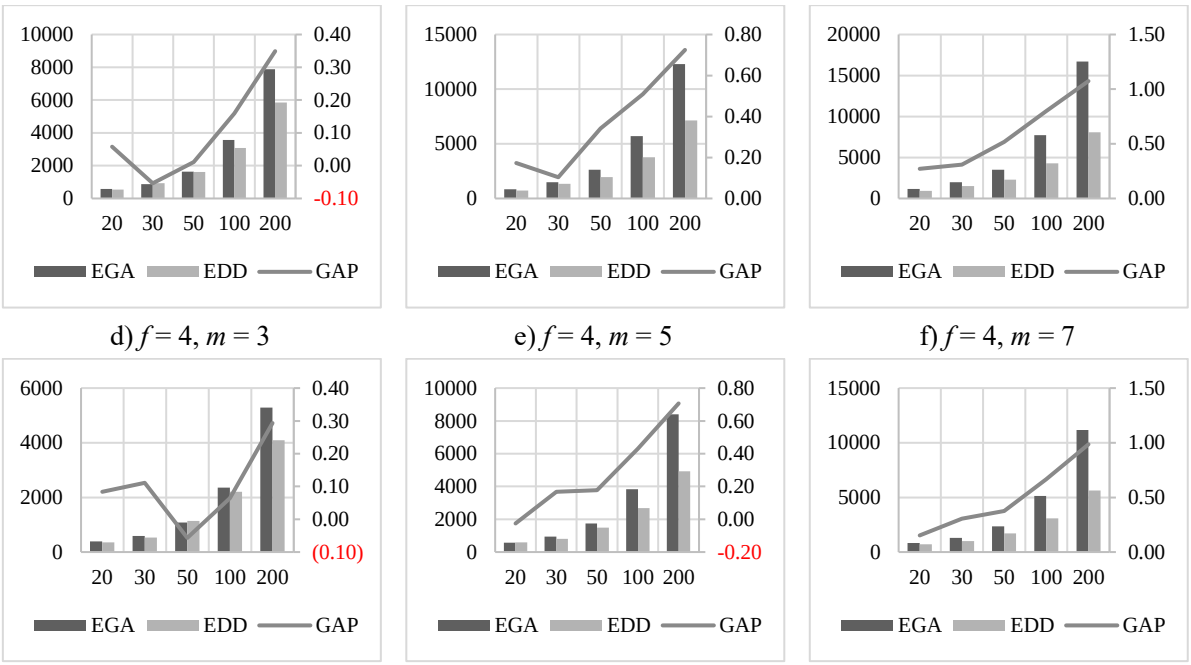
组	观测数	求和	平均	方差
EGA_OS	54	18.14698	0.336055	0.097922
DDE	54	0.154412	0.002859	0.000136



a) $f = 2, m = 3$　　b) $f = 2, m = 5$　　c) $f = 2, m = 7$

图 4.3　DDE 与 EGA_OS 算法的对比结果图

Fig. 4.3　Computational results between DDE and EGA_OS



d) $f = 4, m = 3$　　e) $f = 4, m = 5$　　f) $f = 4, m = 7$

g) $f = 6, m = 3$　　h) $f = 6, m = 5$　　i) $f = 6, m = 7$

图 4.3　DDE 与 EGA_OS 算法的对比结果图（续）

4.6　本章小结

本章研究了目标为极小化双代理加权最大完工时间，考虑释放时间的分布式开放作业调度问题。建立了混合整数规划模型来描述这一 NP 难问题。提出了 EAF-ADA-DS 启发式算法对大规模算例进行求解，设计了考虑问题特性的下界，通过数值实验验证了启发式算法的收敛性。同时，提出了有效的差分进化算法来求解中小规模问题，通过随机可用规则与启发式算法并存的初始化策略来平衡初始解的质量与广度。最后，通过数值仿真实验进行了测试，证明了本文提出算法的优越性。

5 结论与展望

本文主要研究了两个带有释放时间的分布式作业调度问题：一是考虑释放时间与处理单元速度的分布式置换流水作业问题，目标为极小化最大完工时间；二是考虑释放时间的分布式双代理开放作业问题，目标为极小化两个代理的加权完工时间和。在制造与服务过程中，极小化最大完工时间指标可以提高服务效率、减少机器负载，极小化最大完工时间和指标可以提高客户满意度，体现客户重要性。针对上述问题，本文建立了相应的混合整数规划模型，并提出了近似算法用于求解不同规模的问题实例，其具体内容如下：

(1) 针对目标为极小化最大完工时间，考虑释放时间与处理单元速度的分布式置换流水作业问题，首先建立了混合整数规划模型，用于 CPLEX 优化软件的求解。为了在小规模情况下快速获得精确解，本文提出了一种分支定界算法，精心设计了剪枝规则与下界来缩小解空间加速算法。通过与 CPLEX 求解器的数值实验对比，验证了所提出精确算法的高效性能。对于中等规模算例，本文设计了一种离散人工蜂群算法，在算法中加入了随机可用规则和接受准则等策略来提高算法性能。通过数值对比实验证明了所提出算法的杰出性能。考虑到最长处理时间优先规则对极小化最大完工时间指标以及流水作业问题较为有效，本文设计了 DLPV-FM 启发式算法用于求解大规模案例。数值仿真实验证明了 DLPV-FM 启发式在求解本问题上具有收敛性。

(2) 针对目标为极小化双代理加权完工时间和，考虑释放时间的分布式双代理开放作业问题，首先建立了混合整数规划模型，用于 CPLEX 优化软件的求解。考虑到优势代理优先规则和稠密算法分别对于双代理问题与开放作业调度问题较为有效，本文设计了 EAF-ADA-DS 启发式算法用于求解大规模案例。根据问题性质设计了问题下界，并与启发式算法进行了数值仿真实验，实验结果证明了 EAF-ADA-DS 启发式在求解本问题上具有收敛性。为了获得更优质的调度结果，本文设计了差分进化算法来深入搜索优解，采用了特殊的编码与解码方式，将最早可用处理单元的分配规则引入了解码过程，并在初始化阶段通过随机可用规则和启发式算法来平衡解的质量与多样性。最后，通过与其他算法的对比证明了本文提出算法的优越性。

在未来的研究方面，对于分布式作业系统调度问题仍有许多需要考虑的地方，首先是相关的约束，阻塞、装配、维修等因素也会对调度产生影响；其次是问题指标，完工时间和、总延误等单目标与多目标情况在生产生活中也具有重要意义；最后是释放时间，未来对于释放时间的研究可以考虑更加动态的情况来更加符合实际。此外，目前模型与算法研究主要为理论研究，未来会基于模型进一步研究实际问题。

参 考 文 献

[1] 李伯虎, 张霖, 王时龙, 陶飞, 曹军威, 姜晓丹, 宋晓, 柴旭东. 云制造——面向服务的网络化制造新模式[J]. 计算机集成制造系统, 2010, 16(01): 1-7+16.

[2] 白丹宇. 流水车间与开放车间调度算法渐近分析[M]. 清华大学出版社, 2015.

[3] Graham R L et al. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey[J]. Annals of discrete mathematics, 1979, 5: 287-326

[4] Johnson S M. Discussion: Sequencing n Jobs on Two Machines with Arbitrary Time Lags[J]. Management Science, 1959, 5(3):299-303.

[5] Garey M R, Johnson D S, Sethi R. The Complexity of Flowshop and Jobshop Scheduling[J]. Mathematics of Operations Research, 1976, 1(2):117-129.

[6] Lenstra J K, Kan A H G R, Brucker P. Complexity of Machine Scheduling Problems[J]. Annals of Discrete Mathematics, 1977, 1:343-362.

[7] Campbell H G, Dudek R A, Smith M L. Heuristic algorithm for N-job, M-machine sequencing problem[J]. Management Science Series B—Application, 1970, 16 (10): B630-B63.

[8] Nawaz M, Enscore Jr E E, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem[J]. Omega, 1983, 11(1): 91-95.

[9] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics[J]. European Journal of Operational Research, 2005, 165 (2): 479-494.

[10] Naderi B, Ruiz R. The distributed permutation flowshop scheduling problem [J]. Computers & Operations Research, 2010, 37: 754-768.

[11] Gao J, Chen R. An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems [J]. Scientific Research and Essays, 2011, 6: 3094-3100.

[12] Gao J, Chen R, Deng W, et al. Solving multi-factory flowshop problems with a novel variable neighbourhood descent algorithm [J]. Journal of Computational Information Systems, 2012, 8: 2025-2032.

[13] Pan Q K, Gao L, Wang L, Liang J, Li X Y. Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem[J]. Expert Systems With Applications, 2019, 124.

[14] Liu J, Reeves C R. Constructive and composite heuristic solutions to the P//Σ Ci scheduling problem[J]. European Journal of Operational Research, 2001, 132(2): 439-452.

[15] Hatami S, Ruiz R, Andrés-Romano C. Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times[J]. International Journal of Production Economics, 2015, 169: 76-88.

[16] Ruiz R, Pan Q K, Bahman Naderi. Iterated Greedy methods for the distributed permutation flowshop scheduling problem[J]. Omega, 2019, 83: 213-222.

[17] Jing X L, Pan Q K, Gao L, et al. An effective Iterated Greedy algorithm for the distributed permutation flowshop scheduling with due windows[J]. Applied soft computing, 2020, 96: 106629.

[18] Lu C, Gao L, Yi J, et al. Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in China[J]. IEEE Transactions on Industrial Informatics, 2020, 17(10): 6687-6696.

[19] Huang J P, Pan Q K, Gao L. An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times[J]. Swarm and Evolutionary Computation, 2020, 59: 100742.

[20] Mao J, Pan Q, Miao Z, Gao L. An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance[J]. EXPERT SYSTEMS WITH APPLICATIONS, 2021, 169: 0957-4174.

[21] 钱斌, 刘获飞, 胡蓉, 等. 混合迭代贪婪算法求解准时生产分布式流水线调度问题[J]. 控制与决策,2022,37(11):3042-3051. DOI:10.13195/j.kzyjc.2021.0426.

[22] Jing X L, Pan Q K, Gao L. Local search-based metaheuristics for the robust distributed permutation flowshop problem[J]. Applied Soft Computing, 2021, 105: 107247.

[23] Li J, Bai S C, Duan P, et al. An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system[J]. International Journal of Production Research, 2019, 57(22): 6922-6942.

[24] 李浩然, 高亮, 李新宇. 基于离散人工蜂群算法的多目标分布式异构零等待流水车间调度方法[J]. 机械工程学报,2023,59(02):291-306.

[25] Li J Q, Song M X, Wang L, et al. Hybrid artificial bee colony algorithm for a parallel batching distributed flow-shop problem with deteriorating jobs[J]. IEEE transactions on cybernetics, 2019, 50(6): 2425-2439.

[26] Wang G, Gao L, Li X, et al. Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm[J]. Swarm and Evolutionary Computation, 2020, 57: 100716.

[27] Wang G, Li X, Gao L, et al. An effective multi-objective whale swarm algorithm for energy-efficient scheduling of distributed welding flow shop[J]. Annals of Operations Research, 2022, 310(1): 223-255.

[28] Lei D, Wang T. Solving distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memeplex grouping[J]. Engineering Optimization, 2020, 52(9): 1461-1474.

[29] 夏霖辉, 吴瑶, 周学良等. 考虑运输时间的分布式置换流水车间调度灰狼优化算法[J]. 湖北汽车工业学院学报,2022,36(04):68-72+80.

[30] 唐红涛, 刘家毅. 改进的布谷鸟算法求解考虑运输时间的分布式柔性流水车间调度问题[J]. 运筹与管理,2021,30(11):76-83.

[31] Chang H C, Liu T K. Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms[J]. Journal of Intelligent Manufacturing, 2017, 28(8): 1973-1986.

[32] Zhang X, Li X T, Yin M H. An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem[J]. International Journal of Bio-Inspired Computation, 2020, 15(2): 113-124.

[33] Zhang G, Xing K. Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion[J]. Computers & Operations Research, 2019, 108: 33-43.

[34] Wang S, Wang L, Liu M, et al. An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem[J]. International Journal of Production Economics, 2013, 145(1): 387-396.
[35] Xu Y, Wang L, Wang S, et al. An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem[J]. Engineering Optimization, 2014, 46(9): 1269-1283.
[36] Hamzadayı A. An effective benders decomposition algorithm for solving the distributed permutation flowshop scheduling problem[J]. Computers and Operations Research, 2020, 123.
[37] Pan Q K, Gao L, Wang L. An effective cooperative co-evolutionary algorithm for distributed flowshop group scheduling problems[J]. IEEE Transactions on Cybernetics, 2022, 52(7):5999-6012.
[38] Wang J J , Wang L . A Cooperative Memetic Algorithm With Learning-Based Agent for Energy-Aware Distributed Hybrid Flow-Shop Scheduling[J]. IEEE Transactions on Evolutionary Computation, 2022, 26(3): 461-475.
[39] Gmys J, Mezmaz M, Melab N, et al. A computationally efficient Branch-and-Bound algorithm for the permutation flow-shop scheduling problem[J]. European Journal of Operational Research, 2020, 284(3): 814-833.
[40] Bai D, Liang J, Liu B, et al. Permutation flow shop scheduling problem to minimize nonlinear objective function with release dates[J]. Computers & Industrial Engineering, 2017, 112: 336-347.
[41] Pinedo M L. Scheduling: Theory, algorithms, and systems[M]. Berlin: Springer, 2012.
[42] Gonzalez T, Sahni S. Open Shop Scheduling to Minimize Finish Time[J]. Journal of the ACM, 1976, 23(4): 665-679.
[43] Bárány I, Fiala T. Nearly optimum solution of multimachine scheduling problems[J]. Szigma, 1982, 15: 177-191.
[44] Chen R. Dense schedules for open-shop with jobs release dates[J]. OR Transactions, 2003, 7: 73-77.
[45] Lawler E L, Lenstra J K, Kan A H G R. Minimizing Maximum Lateness in a Two-Machine Open Shop[J]. Mathematics of Operations Research, 1981, 6(1):153-158.
[46] Chen R J, Huang W Z, Tang G C. Dense open-shop schedules with release times[J]. Theoretical Computer Science, 2008, 407: 389-399.
[47] Louis S J, Xu Z. Genetic algorithms for open shop scheduling and re-scheduling[C]//Proc. of the 11th ISCA Int. Conf. on Computers and their Applications. 1996, 28: 99-102.
[48] Liaw C F. A hybrid genetic algorithm for the open shop scheduling problem[J]. European Journal of Operational Research, 2000, 124(1): 28-42.
[49] Rahmani Hosseinabadi A A, Vahidi J, Saemi B, et al. Extended genetic algorithm for solving open-shop scheduling problem[J]. Soft computing, 2019, 23: 5099-5116.
[50] Hoogeveen J A, van de Velde S L. Minimizing total completion time and maximum cost simultaneously is solvable in polynomial time[J]. Operations Research Letters, 1995, 17(5): 205-208.
[51] Agnetis A, Mirchandani P B, Pacciarelli D, Pacifici A. Scheduling problems with two competing agents[J]. Operations Research, 2004, 52(2): 229–242.
[52] Agnetis A, Pascale G D, Pacciarelli D. A Lagrangian approach to single-machine scheduling problems with two competing agents[J]. Journal of Scheduling, 2009, 12 (4): 401-415.
[53] Ng C T, Cheng C T E, Yuan J J. A note on the complexity of the two-agent scheduling on a single machine[J]. Journal of Combinatorial optimization, 2006, 12: 387-394.

[54] 梁建恒, 薛含钰, 白丹宇, 苗蕴慧. 分支定界算法求解带有释放时间的单机双代理调度问题[J]. 运筹与管理, 2019, 28(10), 83-88.

[55] Mor B, Mosheiov G. Scheduling problems with two competing agents to minimize minmax and minsum earliness measures[J]. European Journal of Operational Research, 2010, 206(3): 540-546.

[56] Mor B, Mosheiov G. Single machine batch scheduling with two competing agents to minimize total flowtime[J]. European Journal of Operational Research, 2011, 215(3): 524-531.

[57] Yuan J J, Ng C T, Cheng T C E. Two-agent single-machine scheduling with release dates and preemption to minimize the maximum lateness[J]. J Sched, 2015, 18: 147–153.

[58] Gerstl E, Mosheiov G. Scheduling problems with two competing agents to minimized weighted earliness–tardiness[J]. Computers and Operations Research, 2013, 40(1): 109-116.

[59] Gerstl E. Mosheiov G. Minmax weighted earliness-tardiness with identical processing times and two competing agents[J]. Computers & Industrial Engineering, 2017, 107: 171-177.

[60] Lee W C, Chung Y H. Hu M C. Genetic algorithms for a two-agent single-machine problem with release time[J]. Applied Soft Computing Journal, 2012, 12(11): 3580-3589.

[61] Dover O, Shabtay D. Single machine scheduling with two competing agents, arbitrary release dates and unit processing times[J]. Annals of Operations Research, 2016, 238(1-2): 145-178.

[62] Wu C C, Lee W C, Liou M J. Single-machine scheduling with two competing agents and learning consideration[J]. Information Sciences, 2013, 251: 136-149.

[63] Mor B, Mosheiov G. Polynomial time solutions for scheduling problems on a proportionate flowshop with two competing agents[J]. Journal of the Operational Research Society, 2014, 65: 151-157.

[64] Chen R X, Li S S. Proportionate Flow Shop Scheduling with Two Competing Agents to Minimize Weighted Late Work and Weighted Number of Late Jobs[J]. Asia-Pacific Journal of Operational Research, 2021, 38(02).

[65] Jiang F H, Zhang X G, Bai D Y. Chin Chia Wu. Competitive two-agent scheduling problems to minimize the weighted combination of makespans in a two-machine open shop[J]. Engineering Optimization, 2017, 50(4): 684-697.

[66] Holland J H. Adaption in Natural and Artificial Systems[M]. University of Michigan Press, Ann Arbor, 1975.

[67] Karaboga D. An idea based on honey bee swarm for numerical optimization[R]. Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[68] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem[J]. European journal of operational research, 2007, 177(3): 2033-2049.

[69] Storn, R, Price, K. Differential Evolution—A Simple and Efficient Heuristic for global Optimization over Continuous Spaces[J]. Journal of Global Optimization, 1997, 11(4):341-359.

[70] Gonzalez T, Ibarra O H, Sahni S. Bounds for LPT schedules on uniform processors[J]. SIAM journal on Computing, 1977, 6(1): 155-166.

[71] Brah S A, Hunsucker J L. Branch and bound algorithm for the flow shop with multiple processors[J]. European Journal of Operational Research, 1991, 51(1): 88-99.

[72] 东北大学. 基于 SPT 规则的双代理开放车间工件加工排序方法:CN201910859191.X[P]. 2020-12-22.