密级：公开　　中图分类号：TP39

☑全日制　□非全日制

浙江工商大學

硕士学位论文

（专业学位）

论文题目：基于改进蚁群算法的云工作流调度优化

作者姓名：邱模奇

专业学位类别：工程管理硕士

专业学位领域：物流工程与管理

研究方向：调度优化

指导教师：谢毅

提交日期：2023 年 1 月

Dissertation Submitted to Zhejiang Gongshang University for Master's Degree of Management

Cloud workflow scheduling optimization based on improved ant colony algorithm

Author： MoQi Qiu

Major： Logistics Engineering And Management

Supervisor： Yi Xie

Jan. 2023

School of Management and E-Business

Zhejiang Gongshang University

Hangzhou, 310018, P. R. China

摘要

近年来，得益于互联网信息技术蓬勃发展，云计算被广泛使用于大规模数据存储及计算中，为处理海量数据，应用云工作流将数据封装成工作流任务，任务之间存在约束关系，利用云计算服务实现自动化执行。云工作流调度优化问题成为学者们研究的热点问题，经过长时间发展以后，不少学者都提出优化效果较为良好的算法。然而，目前云工作流调度优化仍面临两大挑战：第一，在工作流任务规模较大时，搜索解空间是一项艰巨的任务，耗费大量时间搜索解且质量不高；第二，算法解空间可能是不完备的，无法进行全域搜索。针对以上问题，本文提出一种改进蚁群算法（R_ACO）来优化云工作流调度问题。R_ACO 算法在蚁群算法基础上进行改进，在任务调度顺序编码和虚拟机分配编码都采用了整数编码方式，确保了搜索空间是完备的。同时，对任务调度顺序和虚拟机分配分别设计信息素模型，根据信息素模型进行状态转移从而构造个体解，采用局部更新和全局更新两种方式对信息素进行更新，局部更新增加当代蚁群个体多样性，全局更新则是对精英蚁群的轨迹进行信息素加强。为提高算法搜索效率，采用两阶段进化策略分别对算法进行迭代进化，同时加入避免传输和负载均衡算法改进个体，增加邻域搜索能力。最后对 R_ACO 算法进行了实验分析，与 HEFT、PEFT、HGA、GALCS、HPSO 和 APRE_EDA 算法进行了比较分析，实验结果表明，R_ACO 算法在工作流调度优化问题下搜索解方案能力更优，在相同时间下往往可以找到更优解。

关键词：云计算；云工作流；调度优化；蚁群算法

Abstract

In recent years, thanks to the vigorous development of Internet information technology, cloud computing has been widely used in large-scale data storage and calculation. In order to deal with massive data, cloud workflow is applied to encapsulate data into workflow tasks. There is a constraint relationship between tasks, and cloud computing services are used to realize automatic execution. Cloud workflow scheduling optimization has become a hot topic for scholars. After a long period of development, many scholars have proposed algorithms with good optimization effect. However, cloud workflow scheduling optimization still faces two major challenges. First, when workflow tasks are large in scale, searching solution space is a difficult task, which consumes a lot of time and has low quality. Secondly, the solution space of the algorithm may be incomplete, and it cannot be searched globally. To solve the above problems, this paper proposes an improved ant colony algorithm (R_ACO) to optimize cloud workflow scheduling. R_ACO algorithm is improved on the basis of ant colony algorithm, and adopts integer coding in task scheduling sequence coding and virtual machine assignment coding, which ensures that the search space is complete. At the same time, pheromone models were designed for task scheduling sequence and virtual machine assignment respectively, and individual solutions were constructed by state transition based on the pheromone models. Pheromones were updated by local update and global update. Local update increased individual diversity of contemporary ant colonies, while global update enhanced pheromone trajectories of elite ant colonies. In order to improve the search efficiency of the algorithm, a two-stage evolutionary strategy was adopted to carry out iterative evolution of the algorithm respectively, and at the same time, transfer avoidance and load balancing algorithms were added to improve the individual and increase the neighborhood search ability. Finally, the R_ACO algorithm is analyzed and compared with HEFT, PEFT, HGA, GALCS, HPSO and APRE_EDA algorithms. The experimental results show that R_ACO algorithm has better ability to search solutions under workflow scheduling optimization problems,

and can often find better solutions under the same time.

Keyword: Cloud computing; Cloud workflow; Scheduling optimization; Ant colony

目录

第1章 绪论

1.1 研究背景及意义

近年来， 随着互联网蓬勃的发展和信息技术设备的普及，全球互联网用户在互联网平台产生的数据量呈现指数增长，这些数据存储以及处理对计算能力提出了高要求，为了存储以及处理海量数据，需要采用计算能力更强的计算范式进行处理数据，云计算应运而生。

云计算是在网格计算(Grid Computing)、并行计算 (Parallel Computing)、效用计算(Utility Computing)以及分布式计算(Distributed Computing) 等基础上发展起来的[1]。云计算作为大规模计算平台由云计算服务提供者和云计算用户组成，云计算服务需求方即用户，随时随地利用互联网按照需求向云计算提供商获取计算资源，而云计算服务提供商采用一种“即付即用”的收费方式，便捷地向用户提供软硬件资源[2]。这种服务方式深受大众喜爱，个人、企业以及组织对云计算认可度越来越高。在全球范围内，许多知名 IT 公司争先恐后地加入云计算竞争行业。国际知名的云计算平台主要有谷歌的 GCP（Google Cloud Platform）、亚马逊的 AWS（Amazon Web Service）、微软的 Azure、IBM 的 Blue Cloud，而国内知名云计算平台有阿里云、华为云、腾讯云、百度云等。

云计算作为一个兴起的领域，企业和学术界目前较为认可美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）对云计算的定义——云计算是一种用户按需付费的服务模式，它提供便利的、泛在的计算资源，如服务器、存储空间、软件应用程序、互联网服务，在共享资源池中的资源可按需配置，通过最小的管理工作即可快速获取和释放资源[3]。云计算具有以下几个特征，①按需自助服务 ②广泛的网络接入③共享资源池④资源弹性服务⑤可评测量化的服务，云计算用户可随时随地利用互联网获得计算资源与服务[3]。云计算按照不同服务类型可分为以下几种：⑴基础设施即服务（Infrastructure as a Service，IaaS）:基础设施即服务是指将基础设施以服务的方式提供给客户，基础设施包括网络、存储和一些基本的计算资源。例如：Amazon EC2.⑵平台即服务（Platform as a Service，PaaS）:平台即服务是指将客户开发的应用程序搭建在提供商的云计算基础设施上，客户不需要过多关注基

础设施。例如：Google App Engine、Microsoft Azure.⑶软件即服务（Software as a Service，SaaS）:软件即服务是指按照客户需求开发软件作为服务提供使用，客户无需担心基础设施以及软件维护，通过不同设备可访问使用对应软件。例如：Office Web Apps[4].

中国信息通信研究院在 2022 年 7 月发布的《云计算白皮书（2022 年）》中提供的全球云计算市场规模的数据图如图 1.1 所示，2021 年以 IaaS、Paas、SaaS 为代表的全球公有云市场达到 3307 亿美元，增速 32.5%[5].

图 1.1 全球云计算市场规模及增速

我国云计算市场持续高速增长。2021 年中国云计算总体处于快速发展阶段，市场规模达 3229 亿元，较 2020 年增长 54.4%。其中，公有云市场（图 1.2）继续高歌猛进，规模增长 70.8%至 2181 亿元，有望成为未来几年中国云计算市场增长主要动力；与此同时，私有云市场突破千亿大关，同比增长 28.7%至 1048 亿元[5]。

工作流源于办公自动化领域，由多个任务协同完成，这些任务通常数据量庞大且复杂，任务无法拆分且需要连续执行，任务之间存在优先级关系，通常由处理机器自动化完成。工作流在现代物流仓储管理中有着广泛应用，仓库已经成为企业的物流中心，通常包括流通加工、库存管理、运输和配送活动。仓储核心业务可分为入库管理、仓储管理和出库作业，而业务也可细分为更具体

的环节[5]。针对仓储管理的工作流调度中，仓储工作流任务是指仓储活动中的工单，包括入库单、出库单、中转单等，任务的规则由仓储需求确定[7]。工作流包含多种任务及子任务，不同任务都由对应的处理机器执行。仓储业务抽象成工作流任务后，依托计算服务对工作流进行自动化执行，从而提高仓储作业效率和质量。云计算作为一种按需供给、按量付费的计算服务范式，其具备高延伸、低成本的特点，为工作流合理调度提供了可能解决方案。云计算环境下工作流调度是一种非常典型的调度问题，它是把具有时序约束的子任务分配给不同的虚拟机资源以满足目标需求[8]。

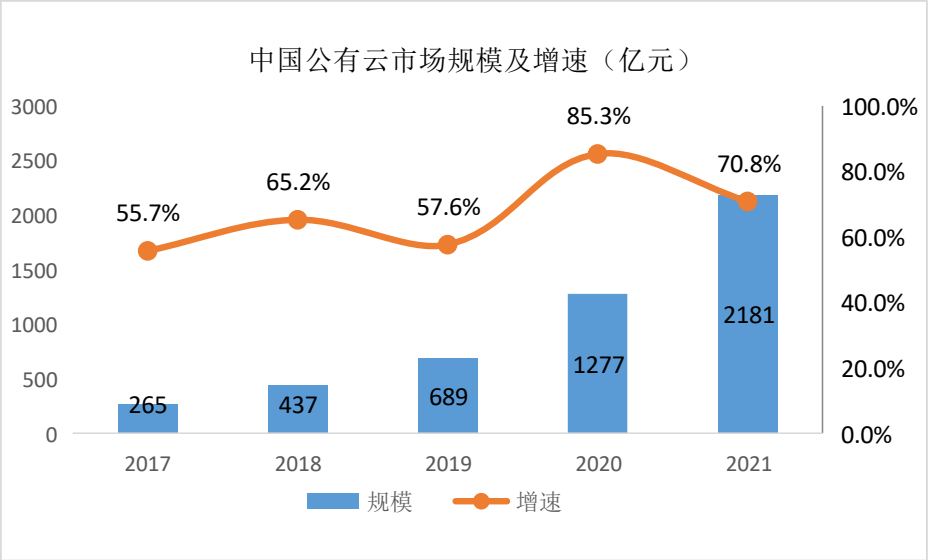

图 1.2 中国公有云市场规模及增速

云工作流在科学计算、电子商务、供应链管理和信息安全等需要大规模存储和高效计算性能的领域有着广泛的应用[9]。科学计算、电子商务、供应链管理等领域在互联网高速发展的前提下，工作流应用往往存在大量并发的任务数据，使得在云计算环境下有着较高的计算能力及存储需求。云工作流调度问题已经被证明是 NP 难问题[10]，通常很难找到最优解。云环境下工作流调度问题研究目的是提高云计算服务质量，通常是最大完工时间或者截止日期约束下的最小成本为目标。由于云用户可随时动态地获取或释放计算资源，导致云环境下任务调度变得尤为复杂。目前针对云工作流调度已经有不少学者研究，但是工作流调度依旧是云计算领域的热点及难点问题。本文是对云环境下面向完工时间的工作流调度进行研究，并做出算法优化。

1.2 国内外研究现状

云计算环境下工作流调度算法多数在异构环境中或同构环境对工作流进行调度的算法。国内外研究人员对云工作流调度展开了研究，也提出了很多有效的算法。这些算法可以分为三类：启发式算法、元启发式算法以及启发式和元启发式混合算法。

启发式算法依赖于特定问题，通常无法保证解是最优的，但可根据特定规则在规定时间内求出良好的解，其解是确定的[11]。Topcuouglu 等[12]提出了两种异构计算环境下工作流任务调度优化算法，即异构最早完成时间（HEFT）算法和处理机上的关键路径（CPOP）算法，以同时满足高性能和最小化调度时间。其中，HEFT 算法根据任务的向上排序值生成任务调度顺序，并将其分配给处理器，采用基于插入式的方法使得完成时间最小化。HEFT 是最为经典和流行的启发式算法。Zhang[13]等在不考虑任务之间传输时间前提下，提出一种有效的优先级和相对距离算法，以最小化工作流的任务调度长度，该方法可有效提高虚拟机的利用率和调度性能。Zhou[14]等提出了一种基于模糊优势排序的异构最早完成时间（FDHEFT）算法，以研究 IaaS 云中工作流调度的成本和完工时间联合优化，该算法将模糊优势排序机制与列表调度启发式 HEFT 紧密结合。Faragardi[15]等提出了一种新的资源配置机制和工作流调度算法，称为贪婪资源配置和改进 HEFT（GRP-HEFT），该算法基于小时成本模型在给定成本预算约束下最小化工作流完成时间。Deldari[16]等基于可用的多核处理资源提出了一种在用户定义最后期限下工作流成本优化算法，该算法采用集群化工作流方式租赁更多核数处理资源来减少时间，在截止时间后将其他集群的任务填补空白时间以降低成本。Arabnejad[17]等开发了一种预算截止日期感知调度（BDAS）算法，其满足预算和时间限制，同时引入一种异构实例的可调成本-时间权衡策略。Bala[18]等提出一种改进的 HEFT 算法，该算法可以减少多种影响因素引起的资源服务器之间等待时间，比如资源服务器之间带宽，存储因素等。Lin[19]等基于弹性计算资源提出了 SHEFT 算法并形式化云计算环境模型和针对该环境的科学工作流以最小化工作流完成时间。Rahman[20]等提出了一种基于动态关键路径的网格自适应工作流调度算法，通过在每一步计算工作流任务图中的关键路径节点来动态确定工作流任务到网格资源的有效映射。Abrishami[21]等基于网格

环境下截止日期约束工作流调度算法提出了两种新的算法：IC-PCP 和 IC-PCPD2，IC-PCP 算法计算工作流任务的 PCP 路径任务节点，优先调度无父任务已经调度的任务，选择任务节点在最晚完成时间前能完成调度最便宜的资源进行调度，随即更新受影响的任务节点的最早完成时间、最晚完成时间，直到所有任务调度完成。而 IC-PCPD2 的不同之处在于为可调度父任务节点选择最便宜资源进行调度。Bittencourt[22]等提出了一种截止日期约束下混合云成本优化算法（HCOC），该算法权衡执行成本和完工时间确定工作流任务执行在公有云资源还是私有云资源上。Sahni 等[23]针对公共云中的科学工作流提出了一种系统计算性能可变和截止时间约束下面向成本优化的动态启发式资源配置和任务调度算法。Arabnejad 等[24]基于 HFET 算法结合 OCT 值提出了一种可预测最早完成时间（PEFT）算法。

元启发式算法是相对于最优化算法提出的，相对于启发式算法往往能找到相对更优的调度方案。常见的元启发式算法以仿自然算法为主，如遗传算法、粒子群算法、蚁群算法、人工蜂群算法等。Barrett[25]等提出了一种云工作流调度器以满足用户服务需求，利用遗传算法结合 MDP 生成最优调度。Zhou[26]等提出了一种基于双层搜索策略的蚁群优化算法，引入预执行时间概念，并将信息素设置阈值。Zhu[27]等针对 IaaS 云工作流调度问题提出了一种基于进化多目标优化算法，该算法引入了一种新的编码方案可表示调度顺序、任务实例分配以及实例规范选择。Wang[28]等提出了一种截止日期约束下混合粒子群成本优化算法，该算法设计了一种空闲时隙感知解码方法，将粒子解码为调度方案，并将那些随机产生而导致任务优先级无效的粒子进行修复。Wu[29]等针对云计算系统下的节能调度问题提出了一种多模型分布式估计算法以最小化完工时间和能耗，其中一个概率模型产生任务调度排列，另一个概率模型产生 VSL 分配集合。Wu[30]等提出了一种基于路径重链增强的分布式估计算法以最小化完工时间，该算法开发了一种基于现有列表启发式的混合解码机制，以调整搜索过程中产生的排列，同时设计了基于相对位置的概率模型来描述解的概率分布，此外设计了两种特定的路径重链方式，以局部增强方式增加搜索能力。Wang[31]等提出了一种基于遗传算法的启发式方法（CGA）来实现异构环境下的任务匹配和调度。Meena[32]等基于云计算的异构性以及虚拟机性能变化和启动时间提出

了一种截止期限约束下成本效益遗传算法以最小化总执行成本。Rodriguez 和 Buyya[33]针对 IaaS 云上科学工作流调度问题提出了一种在截止期限约束下基于粒子群优化算法的调度方法以最小化执行成本。Li[34]等基于粒子群优化算法提出了一种针对云科学工作流的安全和成本感知调度算法（SCAS），SCAS 在满足截止日期和风险率约束下最小化工作流执行成本。Akbari[35]等提出了一种异构环境下基于布谷鸟优化算法（MOSCOA）的多目标算法，该算法在减少执行时间的同时也考虑了任务并行执行。Damodaran[36]等针对批处理机（BPM）调度问题提出了一种模拟退火算法（SA），该算法以一组相同的批处理机的处理时间最小化为目标。Akbari[37]等针对异构环境下的静态任务调度问题提出了一种基于遗传算法的调度方法（EGA-TS），该算法在初始种群时嵌入了一个非随机调度的个体，与基本的遗传算法不同。Khajemohammadi[38]等针对网格环境下工作流调度问题提出了一种基于遗传算法的多目标调度方法（LWSGA），该方法将工作流任务分为各个层次可避免任务优先级失效，快速生成解决方案。Verma 和 Kaushal[39]提出了一种基于遗传算法的预算约束下优先级算法，工作流的每个任务都使用底层次值和顶层次值来分配优先级。Xia 等[40]针对完工时间以及能耗两个目标提出了一种改进的遗传算法（GALCS），通过判断当前个体调度顺序与精英个体调度顺序的最长子串，从而减少个体变异以及交叉概率。

启发式算法和元启发式算法各有其优缺点，学者也有将其结合使用以解决云工作流调度问题。Wu[41]等提出了一种多目标进化列表调度算法（MOELS）以同时优化云工作调度的最大完工时间和经济成本，该算法将经典的启发式列表调度算法嵌入多目标进化算法（MOEA）。Wu[42]等针对云工作流调度提出了一种截止日期约束下结合启发式方法 ProLiS 的元启发式算法 L-ACO 以最小化工作流的执行成本。Ahmad[43]等针对异构环境下云工作流调度提出了一种任务调度由启发式列表调度生成及资源分配由遗传算法生成的 HGA 算法，该算法初始化种群时播入一个 HEFT 个体以减少搜索时间，引入了负载均衡确保资源利用率。Keshanchi 等[44]针对云环境静态工作流任务调度提出了一种结合最早完成时间的启发式资源分配方式（EFT）的遗传算法（NGA），该算法高频搜索将子任务分配给处理器。Pandey[45]等针对云环境下工作流调度提出了一种基于粒子群优化算法的启发式算法以最小化工作流执行总成本。曹[46]等提出了一种

截止时间约束下结合关键路径法的粒子群优化算法以最小化工作流成本。Verma 和 Kaushal[47]针对 IaaS 云上工作流调度提出了一种基于非支配排序的多目标混合粒子群优化算法（HPSO），该算法结合了基于列表的启发式算法，在截止日期和预算约束下，研究分析了完工时间、总成本和能耗之间的关系。Goncalves[48]提出了一种结合列表启发式方法的混合随机键遗传算法来解决家纺工业中常见的生产和切割问题。Wen[49]等针对异构多处理器系统任务调度提出了一种结合遗传算法、变邻域搜索算法和传统列表调度启发式算法的调度方法。Zhang[50]等针对云环境下科学工作流调度提出了一种基于自适应惩罚函数的多目标进化算法。Xu[51][52]等针对异构环境下任务调度提出了一种结合了启发式方法的混合化学反应优化算法（HCRO）和一种结合了最早完成时间启发式算法的多优先级队列遗传算法（MPQGA）来最小化工作流完工时间。以上启发式算法与元启发式算法相结合的算法由于资源分配或任务调度顺序之一是由启发式算法生成的，故其搜索空间是不完备的，存在找不到最优解的可能。

启发式算法通常可以在短时间内找到较不错的调度方案，但其算法性能随着工作流不同而变化。元启发式算法理论上是能搜索到最优解，但其搜索时间往往很长，搜索效率通常不高。综合以上因素，本文提出一种改进蚁群算法来求解云工作流调度问题。本文提出的算法结合启发式算法和蚁群算法，可以进行全域搜索以找到最优解。

1.3 论文主要工作及创新点

本文通过梳理、对比分析国内外专家学者们的研究成果，发现在云工作流调度领域的优化算法可能存在如下局限性：

第一，在工作流任务规模较大时，算法复杂度提高，从而耗费更多时间进行搜索解，通常解质量不高。

第二，算法解空间不完备，存在找不到最优解的可能。

针对以上问题，本文在云工作流领域学者研究基础上，提出了一种改进蚁群算法（R_ACO），并采用 C++语言实现算法。然后，选用大众熟知且认可的科学工作流案例与现有较为经典的算法（HEFT[12]、PEFT[38]、HGA[43]、

GALCS[40]、HPSO[47]、APRE_EDA[30]）进行了对比实验，实验结果验证了本文提出算法的有效性。相较于现在的一些算法，本文提出的 R_ACO 主要创新点如下：

（1）采用基于拓扑排序的整数编码方法，使得任意一个工作流调度方案都存在一个对应的个体，因此最优解一定存在于其搜索范围内，其搜索范围是完备的。

（2）蚁群初始化时，播入一个 HEFT 或 PEFT（取两者中较优）蚂蚁个体，使得算法在较好解开始搜索。

（3）在信息素状态转移时，在启发信息计算时考虑资源负载，算法更契合负载均衡思想；采用局部更新信息素与全局更新信息素结合的方式，在当代蚁群生成个体时防止陷入局部最优，算法收敛速度更快。

（4）采用两阶段进化策略，第一阶段在生成个体时使用启发式算法生成个体虚拟机分配方案，加速算法收敛；第二阶段设计了两种改进策略，进行邻域搜索。

1.4 论文结构

本文主要研究的是云工作流调度优化问题，论文的结构安排如下：

第 1 章 绪论：

本章介绍了云工作流调度的研究背景和选题意义，对国内外学者的研究现状进行了综述和分析，并梳理了当前研究成果的局限性，明确本文的研究内容、创新点和论文结构安排。

第 2 章 相关理论及模型定义：

本章节围绕研究内容，介绍了云工作流相关理论概述，并详细描述了云工作流过程模型、资源模型以及任务执行时间计算模型，给出了云工作流调度优化模型，最后设计了一个小规模案例调度甘特图实验。

第 3 章 基于改进蚁群算法的云工作流调度优化

本章是提出的基于改进蚁群算法具体过程，首先介绍了蚁群算法原理并分析

其应用场景，随后详细说明了提出的R_ACO算法内容，包括个体编码以及解码、信息素模型构造以及更新机制、状态转移概率规则以及构造个体解、改进策略，并给出了具体算法伪代码。最后给出了R_ACO算法整体框架并分析了算法时间复杂度。

第4章 实验评价

本章是实验环节，首先介绍了实验中用到的案例，并确立资源（虚拟机）配置、参数设置、程序运行的硬件环境以及改进蚁群算法的终止条件，在不同规模不同类型的案例下，对比分析了改进蚁群算法与部分现有算法的实验结果，验证了改进蚁群算法的有效性。

第5章 总结与展望

本章是总结本文主要完成的工作，指出论文存在的不足以及进一步的研究方向。

1.5 本章小结

本章首先给出了云工作流调度的研究背景和意义，然后梳理、分析国内外研究现状，并分析现有研究算法的局限性，随后介绍了本文的主要研究内容以及创新点。

第2章 相关理论及模型定义

2.1 云工作流相关概述

云计算是基于分布式计算、大数据、共享资源的一种新型计算范式[53]。随着互联网发展迅速，以科学研究和商业领域为代表的行业对云计算的需求都与日俱增，物流和电子商务行业同样产生海量数据。随着大数据时代到来，数据除了需要储存空间，也需要更多的资源来处理。传统的工作流系统面对高速发展且数据量庞大的工作流需求显得力有不逮，而近些年发展迅速的云计算能很好的处理工作流问题。工作流是一种业务过程抽象成任务自动化完成的计算模型。常见的是以计算密集和数据密集为主的科学工作流和以实例密集为主的商业工作流[54]。

云工作流是工作流管理系统在云环境下的一种新型应用模式[55]。集成工作流系统与云计算，云工作流使得资源获取与释放、管理监控变得快捷方便。云工作流一般具有透明性、可伸缩性以及实时监控特点，透明性是指云工作流可提供运行时，任务调度、负载均衡以及自配置机制；可伸缩性是说明云工作流可根据用户需求实现资源自动配置，可避免资源不够以及资源浪费现象；实时监控说明可通过侦测功能，了解掌握运行情况，及时应对故障发生[56]。

云工作流的核心问题是任务调度问题，即工作流中的实例被拆分为许多有依赖关系的不可拆分任务，然后将这些任务分配给合理的资源上处理。云工作流调度的目的不是简单地提供给用户一个调度方案，仅仅保证工作流顺利完成[57]。云工作流调度考虑更多是 QoS 即服务质量，满足用户 QoS 需求。云计算服务是一种即付即用的模式，也就意味着用户按照使用时间付费，合理的调度能减少云资源使用时间以减少费用；同时也可以释放云资源，使得资源利用率提高。云工作流调度已被证明是一个 NP-hard 问题，通常不能在多项式时间内求出最优解。在环境异构以及资源复杂的情况下，云工作流调度过程十分复杂，常见的是用启发式算法和元启发式算法求解。启发式算法通常依赖特定工作流案例，其求解速度快，质量通常不高；元启发式算法通过迭代进化搜索更优解，其算法复杂度以及搜索空间影响搜索效率。常见的元启发式算法有：蚁群算法、遗传算法、粒子群算法、模拟退火算法等。

2.2 云工作流过程模型和资源模型

云工作流模型可用有向无环图（DAG）表示，定义为$C=\{T,Z\}$,其中$T=\{t_1,t_2,\cdots,t_N\}$代表云工作流模型中任务集合，N是整个工作流任务总数，t_n可表示为任务的编号为n，$size_{t_n}$表示为任务t_n的数据大小，数据对应虚拟机资源执行能力；Z则表示为工作流任务之间约束关系的集合，Z中每个约束关系可表示为$z\left(t_n,t_{\bar{n}}\right)$，$z\left(t_n,t_{\bar{n}}\right)$表示任务$t_n$优先于$t_{\bar{n}}$执行，任务$t_{\bar{n}}$必须在$t_n$被执行完后才能被虚拟机执行，通常也将存在这种约束关系的两个任务称为父子任务，在这里定义$PR_n=\left\{\bar{n}\middle|z_{\bar{n},n}\in Z\right\}$表示任务$t_n$的父任务集合，$SC_n=\left\{\bar{n}\middle|z_{n,\bar{n}}\in Z\right\}$表示任务$t_n$的子任务集合。

任务在执行前后需要传输文件，InF_n表示任务n被虚拟机执行时需要输入的文件集合，$OutF_n$表示任务n被虚拟机执行时后产生的输出文件集合，每个任务都具有输入输出文件，$size_{file}$表示为输入输出文件集合中文件$file$的大小。父子任务之间必然存在至少一个文件$file$，既是父任务的输出文件又是子任务的输入文件，当这个约束关系成立时，两个任务即称为父子任务。父子任务关系进一步延伸，可找寻到任务之间存在祖先子孙关系，如果存在一个任务序列$t_{n^1},t_{n^2},\cdots,t_{n^m}$，满足$t_{n^q}$是$t_{n^{q+1}}$的父任务，其中$1\le q<m$，那么$t_{n^1}$是$t_{n^q}$的祖先任务，$t_{n^q}$是$t_{n^1}$的子孙任务。

云计算环境下存在多种虚拟机资源类型，每个任务同时只能分配给一个虚拟机资源处理，可定义虚拟机资源模型为：$VM=\{vm_1,vm_2\cdots,vm_M\}$，$vm_m$表示虚拟机资源$m$，即编号为$m$的虚拟机资源。$es_m$是$vm_m$的处理能力，$bw_m$是$vm_m$的带宽。文中用到的符号说明如表 2.1 所示：

表 2.1 符号说明表

定义	描述
n	任务编号 n
$size_n$	任务编号 n 的数据大小
m	虚拟机编号
PR_n	任务 t_n 父任务集合
SC_n	任务 t_n 子任务集合
RTS	当前可调度的任务集合
ld_m	虚拟机 m 的负载
lvl_n	任务 t_n 的层次值
pt_n	任务 t_n 的处理时间
s_i	任务 t_i 的开始时间
C_n	任务 t_i 的完成时间
rt_n	任务 t_n 的就绪时间
at_n	任务 t_n 的资源可得时间
$ds_{n',n}$	$t_{n'}$ 和 t_n 之间的数据/文件传输量
ct_n^{in}	数据从共享数据中心传输时间
ct_n^{out}	数据传向共享数据中心时间
$x_{n,m,r}$	决策变量，t_n 在资源 vm_m 上执行顺序为 r
$file$	文件 $file$ 的名称
sz_{file}	文件 $file$ 的大小
InF_n	输入文件集合
$OutF_n$	输出文件集合
ER_n	可以处理任务 t_n 的虚拟机资源集合
ATL_m	虚拟机资源 vm_m 的可得时间段列表
es_m	虚拟机资源 vm_m 的处理能力
bw_m	虚拟机资源 vm_m 的带宽
ET_m	虚拟机资源 vm_m 能够处理的任务集合
ms	工作流完工时间

2.3 云工作流任务执行时间计算模型

工作流通常以文件形式处理数据，传输这些文件的方式会影响调度算法性能，尤其是成本和最大完工时间。一种常见的方式是点对点模型（P2P），另一种方式是使用全局共享存储系统作为文件存储[61]。P2P 模型设定文件直接从处理父任务的虚拟机传输到处理子任务的虚拟机，这意味任务是同步通信的，虚拟机必须保持一直运行，直到子任务都接收完所有数据。这种方式会导致虚拟机租赁成本偏高。此外，虚拟机假如发生故障，会导致数据丢失，并且整个

系统宕机，需要重新任务才可保障调度完整。因此，本文采用共全局共享储存模式进行数据传输。全局共享存储模型中，任务被执行前首先在全局存储中读取文件信息，任务执行完成后将输出文件存储在共享数据中心。特别注意的是，当一个任务与其父任务在同一台虚拟机上执行时，不需要读取其父任务产生的输入文件。

任务 n 中通常会包括文件 $file$，定义 sz_{file} 表示文件的大小。任务 n 传输到子任务 $\bar{n}$ 的中间数据量计算公式如（2-1）所示，从共享数据中心传输到任务 n 的原始数据量计算公式如（2-2）所示。

$$ids_{n',n} = \sum_{file \in InF_n \wedge file \in OutF_{n'}} sz_{file} \tag{2-1}$$

$$ods_n = \sum_{file \in InF_n \wedge file \notin \bigcup_{n' \in PR_n} OutF_{n'}} sz_{file} \tag{2-2}$$

执行任务时，从共享数据中心输入数据（包括父任务产生的中间数据和原始数据）所需时间由公式（2-3）计算出，计算输出数据传输到共享数据中心所需时间为公式（2-4）。

$$ct_n^{in} = \sum_{n' \in PR_n \wedge VM_{n'} \neq VM_n} ids_{n',n} / bw_m + ods_n / bw_m \tag{2-3}$$

$$ct_n^{out} = \sum_{file \in OutF_N} sz_{file} / bw_m \tag{2-4}$$

因此，任务的执行时间计算公式如（2-5）所示：

$$pt_n = size_n / es_m + ct_n^{in} + ct_n^{out} \tag{2-5}$$

2.4 云工作流调度优化模型

面向完工时间的云工作流调度优化数学模型如下：

目标函数：

$$Min\ ms = Max(C_1, \cdots, C_N) \tag{2-6}$$

约束条件：

$$\omega_n = \sum_{m=1}^{M}\sum_{r=1}^{N} x_{n,m,r}\, pt_n^m, \forall n \tag{2-7}$$

$$\sum_{r=1}^{n}\sum_{n=1}^{n} x_{n,m,r} = 1, \forall m \tag{2-8}$$

$$\sum_{n=1}^{N} x_{n,m,r} \le 1, \forall m, r \tag{2-9}$$

$$rt_n = \begin{cases} 0 & PR_n = \varnothing \\ \max\limits_{n' \in PR_n} \{C_{n'}\} & PR_n \neq \varnothing \end{cases} \tag{2-10}$$

$$s_n \ge rt_n, \forall n \tag{2-11}$$

$$C_n = s_n + \omega_n \tag{2-12}$$

$$\sum_{n1=1}^{N} x_{n1,m,r} \ge 1 \sum_{n2=1}^{N} x_{n2,m,r+1}, \forall r \le N-1, m \tag{2-13}$$

$$x_{n,m,r} = 0 \text{ or } 1, \forall n, m, r \tag{2-14}$$

其中：约束（2-7）表示任务在虚拟机上的处理时间；约束（2-8）表示确保每个任务只能被处理一次；约束（2-9）确保对于特定的虚拟机给定位置，该虚拟机为了避免时间冲突仅能同时处理一个任务；约束（2-10）表示当前任务不存在父任务时就绪时间为 0，否则就绪时间为父任务最晚完成时间；约束（2-11）表示任务开始时间只会比任务就绪时间晚或者等于就绪时间；约束（2-12）表示任务完成时间等于任务开始时间加上处理任务的时间；约束（2-13）确保任务在一台虚拟机上第 r+1 个处理的任务的完成时间无法小于第 r 个处理的完成时间；约束（2-14）表示对于决策变量对于任意的任务、虚拟机以及位次只会等于 0 或者 1。

2.5 小规模案例甘特图

云工作流调度问题的本质是获取最优的调度方案，本节设计了一个小规模案例实验，将云工作流调度过程描述更为清楚。本次实验引用了一个小规模案例[62]（如图 2.1 所示），让本文提出的算法 R_ACO 以及对比算法（HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA）分别迭代相同代数，进而对比各自算法下调度甘特图。

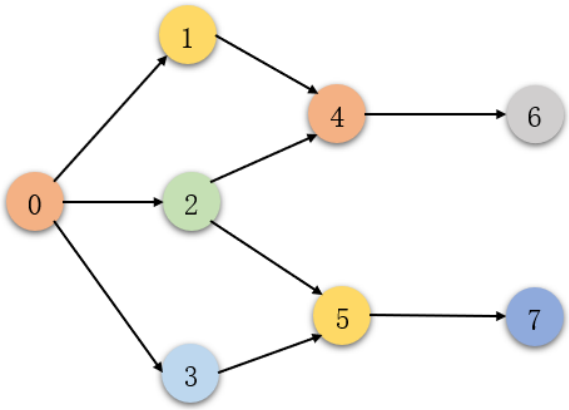

图 2.1 小规模案例 DAG 图

这个案例只有 8 个任务，彼此之间存在约束关系，实验设置两台处理能力不同的虚拟机，将各自运行 100 代的调度结果形成以下甘特图，其中，HEFT、PEFT、HGA、HPSO 以及 APRE_EDA 算法调度方案完全一致，如图 2.2 所示；GALCS 调度方案甘特图如图 2.3 所示；R_ACO 算法调度方案甘特图如图 2.4 所示。

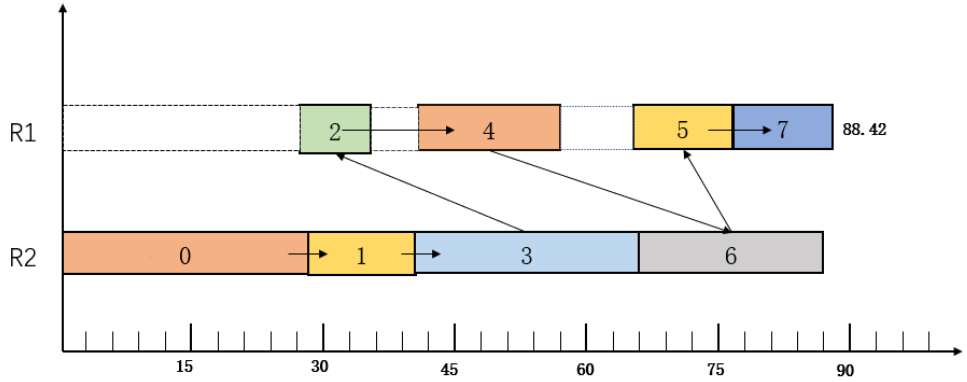

图 2.2 HEFT、PEFT、HGA、HPSO、APRE_EDA 算法调度方案甘特图

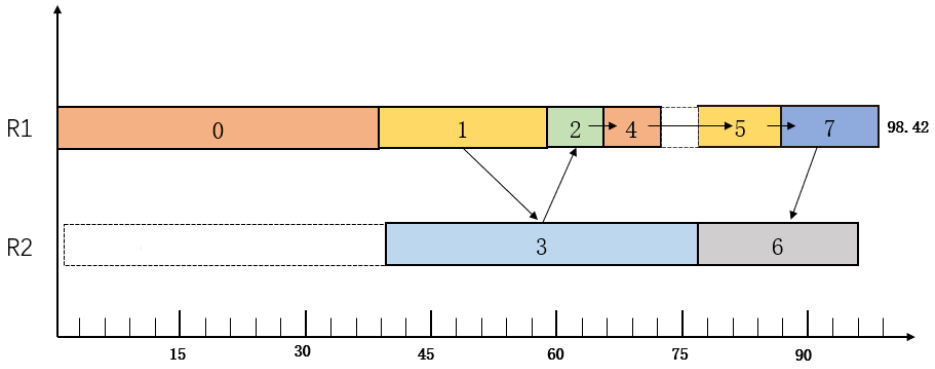

图 2.3 GALCS 算法调度方案甘特图

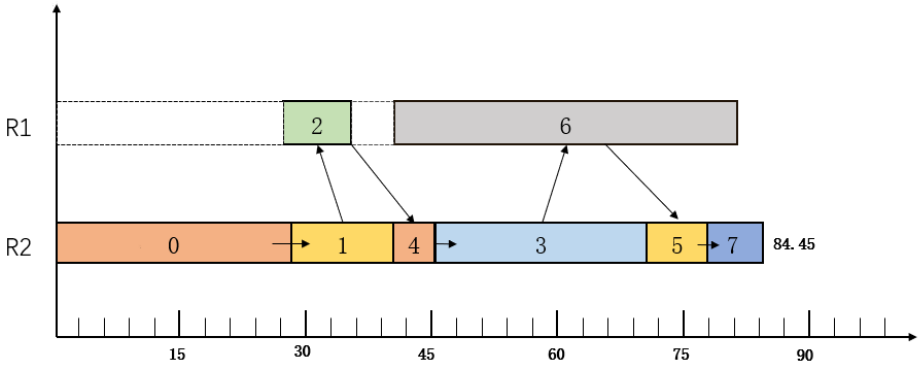

图 2.4 R_ACO 算法调度方案甘特图

对小规模案例调度方案甘特图分析得知，R_ACO 算法在搜索 100 代求出最优方案获得的完工时间为 84.45，而 HEFT、PEFT、HGA、HPSO、APRE_EDA 算法在搜索 100 代求出最优方案获得的完工时间为 88.42，而 GALCS 算法搜索 100 代求出最优方案获得的完工时间为 98.42，R_ACO 算法在调度方案搜索效率上优于其他算法。

2.6 本章小结

本章首先介绍了云工作流相关概念，然后对云工作流过程模型、资源模型以及任务执行时间计算模型进行了详细阐述，最后给出了云工作流调度优化数学模型。

第3章 基于改进蚁群算法的云工作流调度优化方法

3.1 蚁群算法

蚁群算法是人类受到自然界蚂蚁觅食启发而提出一种基于人工蚁群的算法，其模仿大自然真实蚁群觅食机制，属于元启发式算法。20 世纪 90 年代，意大利 M.Dorigo 博士首次提出蚁群算法[58],并将这种搜索算法应用于旅行商问题（TSP）[59]。由于蚁群算法具有良好的搜索效果，且算法鲁棒性强，蚁群算法一经提出即引起了学者广泛讨论及研究。

3.1.1 蚁群算法原理

蚂蚁在觅食活动中通过分工、相互协作寻找到巢穴与食物之间最佳路径，蚂蚁从巢穴出发，向着食物方向可找寻到一条最短路径。不仅如此，当路径出现障碍物时，蚁群也能在一定时间内重新找到最优的路径。当障碍物出现时，蚁群是均匀分布地向不同路径出发，由于蚂蚁爬行速度几乎是一样的，在相同的时间内，较短路径上爬过的蚂蚁数量是大于较长路径，一段时间后，多数蚂蚁更倾向于较短路径上经过。经过大量研究观察发现，蚂蚁自身能分泌一种信息素，并且可识别该路径上信息素浓度，信息素浓度越高的路径，蚂蚁选择爬行的概率则越大。由此，蚂蚁之间可通过分泌信息素以及识别信息素浓度来协作完成觅食最短路径。下面将以图 3.1 为例，介绍蚂蚁应对障碍物时择优路径原理。

假设 M、N 两点分别代表蚁群巢穴和食物所在地，两点之间共有两条路径可到达，分别为：M-Q-R-E-N 和 M-Q-W-E-N,其中 E-W 和 Q-W 的距离都定义为 1.5m，R-E 和 Q-R 的距离则定义为 3m。在 t=0 时刻（图 3.1（b））， 存在 120 只蚂蚁走到分支路口 Q 点或者 E 点，此刻蚂蚁们可选择的方向有两个，而此时两条路径上信息素都是为空，因此蚂蚁会按照相同概率选择两条路径之一爬行，即 60 只蚂蚁选择的路径是 Q-W 或 E-W，另外 60 只蚂蚁选择的路径是 Q-R 或 E-R。如此一来，蚂蚁在边爬行过程中不断分泌信息素。假设定义每只蚂蚁具有相同的爬行速度为 3m/s,每只蚂蚁释放信息素能力相同为 1 强度。

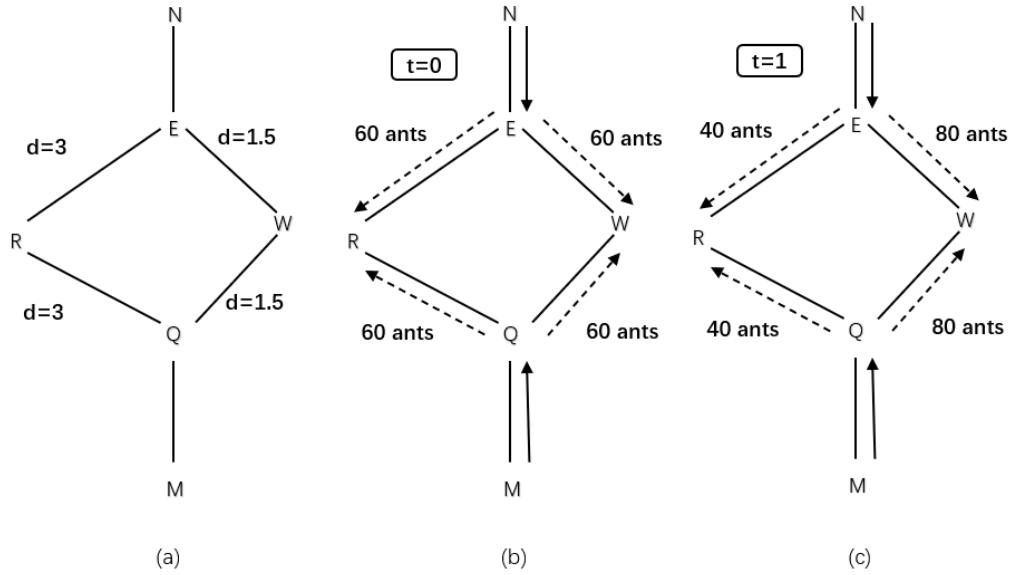

图 3.1 蚂蚁选择最佳路径过程图

那么，经过 1s 以后，从 Q 点出发的 120 只蚂蚁中有 60 只到达 R 点，而另外 60 只经过 W 点到达了 E 点；从 E 点出发的 120 只蚂蚁中有 60 只到达了 R 点，另外 60 只经过了 W 点到达 Q 点。从结论看，Q-R 和 E-R 都只有 60 只蚂蚁经过信息素强度为 60，Q-W 和 E-W 上都有 120 只蚂蚁经过，其信息素强度为 120。这么一来，当有新来的 120 只蚂蚁时，蚂蚁则依据信息素浓度去选择路径。当 t=1 时刻时（图 3.1（c）），因为 Q-W 和 E-W 路径的信息素浓度分别为 Q-R 和 E-R 的两倍，当新来的 120 只蚂蚁走到 Q 点或 E 点时，只有 40 只蚂蚁选择往 Q-R 和 E-R 方向走，另外 80 只蚂蚁都选择信息素浓度更高的 Q-W 和 E-W，蚂蚁数量与信息素浓度成正相关。因此，可以看出路径更短则信息素浓度会更高，蚁群也能更快发现此路径。

3.1.2 蚁群算法在 TSP 问题中的应用

TSP 问题[60]是经典的组合优化问题，是一个 NP-hard 问题。下面将介绍蚁群算法在 TSP 上的应用。假设有一个旅行商人要拜访 n 个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径长度为所有路径之中的最小值。该问题数学

模型可定义为：$G=(N,E)$,其中 $N=\{1,2,\cdots n\}$ 表示城市节点数量，$E=\{(i,j)\mid i,j\in N\}$ 表示城市 i 到城市 j 的路径，$d(i,j)$ 表示城市 i 与城市 j 之间的距离。

基于蚁群算法的 TSP 问题，可用 $\tau_{i,j}(t)$ 表示 t 时刻城市 i 到城市 j 之间信息素浓度，其初始时候每条路径上信息素相等，可定义为 $\tau_{i,j}(0)=C$,其中 C 为常数。$Ant_i(t)$ 表示 t 时刻位于城市 i 的蚂蚁个数，pop 代表蚁群中蚂蚁总数，$p^k{}_{i,j}(t)$ 表示 t 时刻蚂蚁 k 由城市 i 转移到城市 j 的概率，其计算公式为：

$$p^k{}_{i,j}(t)=\begin{cases}\tau_{i,j}(t)^{\alpha}\cdot\eta_{i,j}(t)^{\beta}\Big/\sum\limits_{j\in allowed_k}\left[\tau_{i,j}(t)\right]^{\alpha}\left[\eta_{i,j}(t)\right]^{\beta} & j\in allowed_k\\ 0 & \text{其他}\end{cases}\tag{3-1}$$

其中，允许选择的城市集合 $allowed_k=\{1,2,\cdots,n\}-tabu_k$，$tabu_k(k=1,2,\cdots,pop)$ 表示蚂蚁 k 当前所走过的城市集合，$tabu_k$ 随着蚂蚁爬行进行动态调整。α 表示信息素在蚂蚁选择下个城市当中的重要程度，$\eta_{i,j}$ 表示启发信息，在 TSP 问题中定义 $\eta_{i,j}=1/d_{i,j}$，β 则表示 $\eta_{i,j}$ 在概率转移中重要程度。当蚂蚁遍历完所有城市，即蚂蚁的路径即是问题的解，随后将对信息素进行更新，其计算公式为：

$$\tau_{i,j}(t+1)=(1-\rho)\cdot\tau_{i,j}(t)+\Delta\tau_{i,j}\tag{3-2}$$

其中，$\rho\in(0,1)$ 表示信息素蒸发因子，信息素浓度会随时间而挥发，$\Delta\tau_{i,j}$ 表示信息素浓度增量，其计算公式为：

$$\Delta\tau_{i,j}=\sum_{k=1}^{pop}\Delta\tau^k{}_{i,j}\tag{3-3}$$

$\Delta\tau^k{}_{i,j}$ 表示蚂蚁 k 在本次路径中在城市 i 和城市 j 留下的信息素增量，若蚂蚁 k 经过城市 i 到城市 j，则 $\Delta\tau^k{}_{i,j}=Q/L^k$，其中 Q 为常数，L^k 表示蚂蚁 k 在本次爬行中走过路径长度。

TSP 问题是寻找最佳路径，根据上述建模分析，在初始时生成若干只蚂蚁，在一段时间后，多数蚂蚁会倾向于路径更短的选择，从而获得这个问题的解。蚁群算法在求解 TSP 问题上，效果通常不错，因此可进一步将蚁群算法应用在云工作流调度问题。

3.2 基于改进蚁群算法的云工作流调度优化方法

云工作流调度问题通常采用启发式算法和元启发式算法来进行任务调度以及资源分配，传统的算法如：遗传算法、粒子群算法、模拟退火算法等。传统算法往往存在搜索空间不完备，或是搜索效率不高的问题，因此本文提出的一种基于改进蚁群算法的云工作流调度算法，因采用了整数编码，保证了任意的调度方案都有其对应的编码方案，可以进行全域搜索，理论上可以找到最优解。整体算法流程图如图 3.2 所示。

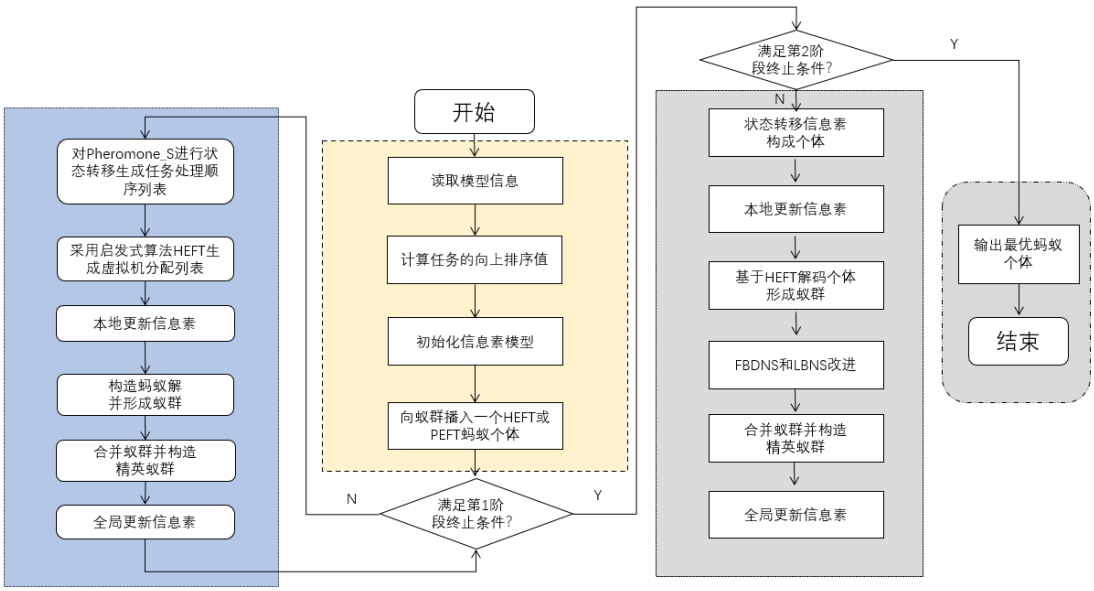

图 3.2　R_ACO 算法流程图

3.2.1 编码方式

对于蚁群算法而言，用编码描述现实问题至关重要，将现实问题与蚂蚁解空间联系在一起便于后续一系列更新改进策略。蚂蚁个体采用整数编码，其方法如下：$Ant=\{task_1,\cdots,task_N,VM_1,\cdots,VM_N\}$，其中 N 为任务数量，其中，$\{VM_1,\cdots,VM_N\}$ 是虚拟机分配列表，VM_m 表示给任务 n 分配的虚拟机编号，即把任务 n 分配给资源 VM_n，$VM_n\in ER_n$，例如：$VM_3=4$ 表示 3 号任务是分配给 4 号虚拟机的；$\{task_1,\cdots,task_N\}$ 是任务调度顺序列表，是所有任务的编号 $(1,\cdots,N)$ 的一个排列，且满足任务的优先级约束，$task_n$ 表示第 n 个被调度的任务的编号，即任务 $task_n$ 是第 n 个被调度的，

例如：$task_6=8$表示第 6 个调度的任务是 8 号任务。当采用任务正向调度时，任何一个任务都必须等待其所有父任务调度完成后（或者本身无父任务）才能开始进行调度，即任意任务出现位置必须在其父任务之后；当采用任务反向调度时，正好与正向调度相反，任何任务出现位置必须在其子任务之后。

任务的层次值计算与其父任务具有直接关系，可根据层次值生成调度顺序。

不存在父任务的任务层次值计算方式为：

$$lvl_n=0 \tag{3-4}$$

在公式（3-4）基础上，其他任务层次值计算方式为：

$$lvl_n=\max_{t_{n^-}\in PR_n}\left\{lvl_{n^-}\right\}+1 \tag{3-5}$$

任务优先级向上排序算法可计算每个任务的$rank$值，在蚁群算法的状态转移概率中，任务调度顺序的启发信息就是任务向上排序值。其计算方式为：

$$rank_n=\overline{\omega_n}+\max_{t_{n^+}\in SC_n}\left\{rank_{n^+}\right\} \tag{3-6}$$

3.2.2 信息素及更新机制

信息素模型

蚁群算法的核心在于信息素模型的构建及更新。在蚁群系统中，蚂蚁爬行经过的轨迹会存在其分泌的信息素，后续而来的蚂蚁可根据信息素浓度选择路径，同时信息素也会随着时间流逝而挥发，一段时间后，多数蚂蚁经过的路径会聚集较多的信息素。在云工作流系统里，可将信息素应用到任务调度顺序和资源分配当中，在算法开始，就会对信息素进行初始化。本文设计的信息素模型包括任务调度顺序信息素模型$Pheromone_S(k)$和资源调度信息素模型$Pheromone_R(k)$，具体构建过程如下：

任务调度顺序信息素模型：

$$Pheromone\_S(k)=\begin{bmatrix}\tau_{1,1}(k) & \cdots & \tau_{1,N}(k)\\ \vdots & \ddots & \vdots\\ \tau_{N,1}(k) & \cdots & \tau_{N,N}(k)\end{bmatrix} \tag{3-7}$$

其中$\tau_{n,n'}(k)$表示在第k代任务t_n之后紧接着调度的是任务$t_{n'}$的信息素。

资源分配信息素模型：

$$Pheromone\_R(k)=\begin{bmatrix}\tau'_{1,1}(k) & \cdots & \tau'_{1,M}(k)\\ \vdots & \ddots & \vdots\\ \tau'_{N,1}(k) & \cdots & \tau'_{N,M}(k)\end{bmatrix} \quad (3\text{-}8)$$

其中 $\tau'_{n,m}(k)$ 表示在第 k 代任务 t_n 分配给资源 vm_m 的信息素。

对于任务调度顺序信息素模型 $Pheromone\_S(k)$ 而言，由于任务之间存在依赖关系，即任务 t_n 无法在其父任务之前调度。因此，定义 $ET=\{t_n|t_n\in N-\xi_n-\zeta_n+SC_n\}$ 是任务 t_n 之后可调度的任务集。其中，N 是所有任务集合，ζ_n 是任务 n 的子孙任务集合，ξ_n 是任务 n 的祖先任务集合，SC_n 是任务 n 的直接子任务集合。定义 $Sum^n_{rank}=\sum_{n\in ET} rank_n$ 表示任务 n 的 ET 中所有任务的 $rank$ 值之和，$\gamma_{n,n'}$ 代表一个标记值：当任务 n 的 ET 中包含任务 n 时，$\gamma_{n,n'}=1$ 否则 $\gamma_{n,n'}=0$，可将 $Pheromone\_S(k)$ 初始化为：

$$Pheromone\_S(1)=\begin{bmatrix}\gamma_{1,1}\times rank_1\Big/\left|Sum^1_{rank}\right| & \cdots & \gamma_{1,N}\times rank_I\Big/\left|Sum^1_{rank}\right|\\ \vdots & \ddots & \vdots\\ \gamma_{N,1}\times rank_1\Big/\left|Sum^N_{rank}\right| & \cdots & \gamma_{N,N}\times rank_I\Big/\left|Sum^N_{rank}\right|\end{bmatrix} \quad (3\text{-}9)$$

由于任务是在异构环境下调度，存在任务 n 无法被虚拟机 m 调度的情况，因此，定义 $\eta_{n,m}=1$ 来表示任务 n 可被虚拟机 m 处理，$\eta_{n,m}=0$ 表示任务 n 无法被虚拟机 m 处理。可将 $Pheromone\_R(k)$ 初始化为：

$$Pheromone\_R(1)=\begin{bmatrix}\eta_{1,1}\Big/ER_1 & \cdots & \eta_{1,M}\Big/ER_1\\ \vdots & \ddots & \vdots\\ \eta_{N,1}\Big/ER_N & \cdots & \eta_{N,M}\Big/ER_N\end{bmatrix} \quad (3\text{-}10)$$

其中 ER_n 表示任务 n 的可调度虚拟机集合。

信息素更新规则

多数蚂蚁经过的路径上信息素浓度通常会高于那些少数蚂蚁经过的路径，

这一机制进一步应用到蚁群算法中称为信息素更新。每一只蚂蚁在完成一次解构造后，即该只蚂蚁先是生成任务调度顺序而后将所有任务都分配给虚拟机完成后，信息素将进行一次更新，称为局部更新。局部更新目的是保持当代蚁群多样性，每生成一只蚂蚁，将其对应任务调度和虚拟机分配上信息素根据规则降低，保证下一只蚂蚁选择同样顺序或分配的概率降低，从而在当代蚁群中生成蚁群更具有多样性，增加了搜索空间，防止陷入局部最优。当代所有蚂蚁都完成解构造后，信息素将进行一次全局更新。

信息素局部更新的计算方法为：

当调度任务 n 后紧接调度 $\bar{n}$ ： $\tau(n,\bar{n})=(1-\rho_1)\times\tau(n,\bar{n})+\rho_1\times\Delta\tau(n,\bar{n})$ （3-11）

其中 ρ_1 为任务调度信息素挥发因子，指的每当蚂蚁迭代一次，任务调度信息素蒸发程度。ρ_1 越大，说明信息素蒸发速率越快；$(1-\rho_1)$ 表示信息素残留程度，其中 $\rho_1\in(0,1]$。

$\Delta\tau(n,\bar{n})$ 表示一只蚂蚁迭代一次时信息素增量，计算公式为：

$$\Delta\tau(n,\bar{n})=\begin{cases}\dfrac{Q}{L^k*N^2} & \text{当任务}n\text{紧接着调度的任务为}\bar{n}\text{时}\\ 0 & \text{其他}\end{cases} \quad (3\text{-}12)$$

其中 Q 为常数，可定义为基于 $HEFT$ 个体的 ms，L^k 是当前蚂蚁调度时间，N 表示任务个数。

当任务分配给虚拟机后： $\tau'(n,m)=(1-\rho_2)\times\tau'(n,m)+\rho_2\times\Delta\tau'(n,m)$ （3-13）

其中 ρ_2 为虚拟机信息素挥发因子，指的每当蚂蚁迭代一次，虚拟机分配信息素蒸发程度。ρ_2 越大，说明信息素蒸发速率越快；$(1-\rho_2)$ 表示信息素残留程度，其中 $\rho_2\in(0,1]$。

$\Delta\tau(n,m)$ 表示一只蚂蚁迭代一次时信息素增量，计算公式为：

$$\Delta\tau(n,m)=\begin{cases}\dfrac{Q}{L^k*M^2} & \text{当任务}n\text{分配给虚拟机为}m\text{时}\\ 0 & \text{其他}\end{cases} \quad (3\text{-}14)$$

局部更新是在每个蚂蚁生成之后就会立即进行的，而全局更新是在一代蚂蚁全部生成并解码完成后进行，依据精英种群的精英个体进行全局更新。精英种群是当代蚂蚁中适应度值最佳的若干个体，可定义为ANT_e，计算精英种群个体数量的方式为$F_e = p_e \times F$，其中F表示种群规模，F_e表示精英种群规模，$p_e \in (0,1)$表示精英率。

信息素全局更新的计算方法为：

当调度任务n后紧接调度$\bar{n}$：

$$\tau(n,\bar{n}) = (1-\rho_1)\times\tau(n,\bar{n}) + \frac{Q}{Ant_{best}} \times \frac{\rho_1}{F_e} \sum_{Ant \in ANT_e} \chi_{n,\bar{n}}^{ant} \tag{3-15}$$

其中，标记值$\chi_{n,\bar{n}}^{ant} = \begin{cases} 1 & \text{当任务} n \text{紧接着调度的任务为} \bar{n} \text{时} \\ 0 & \text{其他} \end{cases}$，$Q$为常数，可定义为基于$HEFT$个体的$ms$，$Ant_{best}$是当代蚁群中最佳蚂蚁调度时间。

当任务n分配给虚拟机m后：

$$\tau(n,m) = (1-\rho_2)\times\tau(n,m) + \frac{Q}{Ant_{best}} \times \frac{\rho_2}{F_e} \sum_{Ant \in ANT_e} \gamma_{n,m}^{ant} \tag{3-16}$$

其中，标记值$\gamma_{n,m}^{ant} = \begin{cases} 1 & \text{当任务} n \text{分配给虚拟机为} m \text{时} \\ 0 & \text{其他} \end{cases}$

3.2.3 状态转移概率及构造个体解

状态转移概率

信息素浓度影响着蚂蚁下一个路径选择，在传统蚁群算法中，将信息素结合启发信息转化成概率来选择下一个节点，启发信息通常与节点间距离成反比。在云工作流调度系统应用下，蚂蚁无论是选择下一个任务还是选择虚拟机处理也必须考虑启发信息，而启发信息选用则会大大影响算法性能。在本文提出的 R_ACO 算法中，将任务调度顺序启发信息定义成任务向上排序值，可表示任务选用的优先级；同时将虚拟机分配启发信息加入了虚拟机负载因素，以此契合负载均衡思想。

任务调度顺序状态转移概率公式为：

$$p_{n,\bar{n}}=\begin{cases}\tau(n,\bar{n})^{\alpha}\cdot\eta(n,\bar{n})^{\beta}/\sum_{n\in RT}\tau(n,\hat{n})^{\alpha}\cdot\eta(n,n)^{\beta} & \hat{n}\in RT\\ 0 & \text{其他}\end{cases} \quad (3\text{-}17)$$

其中，启发信息$\eta(n,\bar{n})=rank_{\bar{n}}$，代表任务$\bar{n}$的向上排序值，$\alpha$和$\beta$分别代表信息素系数和启发信息系数。

虚拟机分配状态转移概率公式为：

$$p'_{n,m}=\begin{cases}\tau(n,m)^{\alpha'}\cdot\eta'(n,m)^{\beta'}/\sum_{m\in ER_n}\tau(n,\bar{m})^{\alpha'}\cdot\eta(n,\bar{m})^{\beta'} & \bar{m}\in ER_n\\ 0 & \text{其他}\end{cases} \quad (3\text{-}18)$$

其中，启发信息$\eta'(n,m)=\frac{1}{ld_m}$，$ld_m$是把当前任务$n$分配给虚拟机$m$后的负载。$\alpha'$和$\beta'$分别代表信息素系数和启发信息系数。

构造个体解

每只蚂蚁除了分泌信息素外，也可识别路径上信息素浓度。之前蚂蚁经过的轨迹影响着后来的蚂蚁，在蚁群算法中也同样蕴含这个道理，每一次算法迭代进化都被受到之前个体影响。因此，构造个体解的方法尤为重要。有效的个体解可促进高质量搜索最优解，相反，劣质的解也会导致无效搜索。本文设计的 R_ACO 算法是基于两阶段进化策略，阶段一是基于任务调度顺序信息素 Pheromone_S 结合启发信息$\eta(n,\bar{n})$再进行状态转移，而后根据转移概率按照伪随机比例规则生成任务调度顺序列表$\{task_1,\cdots,task_N\}$和基于任务最早完成时间生成虚拟机分配列表$\{VM_1,\cdots,VM_N\}$的方法构造个体解，采用基于任务最早完成时间的启发式方法目的是提升解质量，加速算法收敛。阶段二是在阶段一基础上进一步使用信息素及启发信息，在虚拟机分配方式上，采用基于虚拟机分配信息素 Pheromone_R 结合启发信息$\eta(n,m)$再进行状态转移，同样使用伪随机比例规则选择虚拟机分配，进而生成虚拟机分配列表$\{VM_1,\cdots,VM_M\}$。

在本节中设计了以下生成个体的算法，目的是使得个体生成时解的质量相对较优，以加速算法收敛。在任务调度顺序生成过程中，因为本文设计的任务调度

顺序信息素模型是针对任务 n 以及之后紧接调度的任务 $\overline{n}$，因此首个任务并未参与信息素更新以及状态转移。故在信息素模型中，加入首个调度任务的概率模型，并且也伴随信息素更新而更新。根据一号调度任务概率模型采用伪随机比例规则选择第一个任务如算法 3.1 所示；根据任务调度顺序信息素模型采用伪随机比例规则生成任务排列 $\{task_1,\cdots,task_N\}$ 如算法 3.2 所示；基于 HEFT 算法生成个体 $\{VM_1,\cdots,VM_N\}$ 并解码个体如算法 3.3 所示；基于虚拟机分配信息素模型采用轮盘赌法生成虚拟机分配列表 $\{VM_1,\cdots,VM_N\}$ 如算法 3.4 所示。

算法 3.1：根据一号调度任务概率模型采用伪随机比例规则选择第一个任务

Function: *GenerateFisrtTask*(*PrbOfFrsTsk*)

1: 设置一个新集合 S ,将不存在父任务的任务添加进集合 S 中;
2: 初始化 $Ant=\{task_1,\cdots,task_N\}$;
3: $P_{sum}\leftarrow\sum_{n\in S}\rho_n$, $P_n\leftarrow\rho_n/P_{sum}, n\in S$;
4: 生成一个随机数，使得 $\mu\in[0,1)$;
5: IF $\mu<0.2$ THEN
6: 遍历集合 S 中每个任务，找到 P_n 最大的任务 n
7: $task_1\leftarrow n$
8: ELSE
9: 产生一个随机数 $\gamma\in[0,1)$;
10: FOR 每个 $n\in S$ DO
11: $pr\ +=P_n$;
12: IF $\gamma>=pr$ THEN
13: $task_1\leftarrow n;break;$
14: ENDIF
15: ENDFOR
16: RETURN *Ant*

在算法 3.1 中，第 1-2 行是设置一个不存在父任务的集合 S 并且初始化任务调度顺序列表；第 3 行是计算 S 中每个任务 n 经状态转移概率公式求出的被选中的概率 P_n，其中 ρ_n 是由首个任务概率集合乘以任务向上排序值计算出；第 4-15 行整体是采用伪随机比例规则进行选择任务排序，此处，本文定义 $u<0.2$ 时，选用其 ρ_n 值最大的任务，反之，根据轮盘赌法选择任务；第 5-8 行遍历 S 中所有任务，将概率最大值找出且把对应的任务 n 赋值给 $task_1$；第 9-15 行基于

首个任务概率和任务向上排序值采用轮盘赌法进行任务选择；第 16 行返回蚂蚁个体。

算法 3.2：根据任务调度顺序信息素模型采用伪随机比例规则生成任务排序

Function: $GenerateTaskOrder(Pheromone\_S)$

1:设置一个新集合 Y 记录已经调度的任务并设置一个集合 S；

2:将不存在父任务的任务添加进集合 S 中;初始化 $Ant=\{task_1,\cdots,task_N\}$；

3: FOR $l \leftarrow \{2,\cdots,N\}$ DO

4: $P_{sum} \leftarrow \sum_{n\in S}\rho_n$, $P_n \leftarrow \rho_n / P_{sum}$, $n \in S$；

5: 生成一个随机数，使得 $\mu \in [0,1)$；

6: IF $\mu < 0.2$ THEN

7: 遍历集合 S 中每个任务，找到 P_n 最大的任务 n

8: $task_l \leftarrow n$

9: ELSE

10: 产生一个随机数 $\gamma \in [0,1)$;

11: FOR 每个 $n \in S$ DO

12: $pr += P_n$;

13: IF $\gamma >= pr$ THEN

14: $task_l \leftarrow n; break;$

15: ENDIF

16: ENDFOR

17: S 集合删除任务 n； Y 集合插入任务 n；

18: FOR 每个 n' in SC_{task_n} DO

19: IF $PR_{n'}$ in Set Y THEN

20: S 插入删除任务 n'

21: ENDIF

22: ENDFOR

23: ENDIF

24:ENDFOR

25:RETURN Ant

算法 3.1 实际只生成了 $task_1$，后续的任务排序需算法 3.2 完成。在算法 3.2 中，第 1-2 行是设置一个不存在父任务的集合 S 以及一个记录已经调度的任务集合 Y 并且初始化任务调度顺序列表；第 4 行是计算 S 中每个任务经状态转移概率公式求出的被选中的概率 P_n，其中 ρ_n 是由任务调度顺序信息素 $Pheromone\_S$ 乘以任务向上排序值计算出；第 5-23 行整体是采用伪随机比例规则进行选择任务

排序，此处，本文定义$u<0.2$时，选用其ρ_n值最大的任务，反之，根据轮盘赌法选择任务；第 6-9 行遍历S中所有任务，将概率最大值找出且把对应的任务n赋值给$task_n$；第 10-16 行任务调度顺序信息素$Pheromone_S$和任务向上排序值采用轮盘赌法进行任务选择；第 17 行将已经完成调度的任务n从S集合中删除同时将任务n添加进集合Y中；第 18-22 行为更新集合S，将那些父任务已经全部调度完的任务添加进集合S；第 24 行返回个蚂蚁个体。

算法 3.3：基于 HEFT 算法生成个体并解码个体

Function: *DecodeForMsByHEFT*(*Ant*)

1:初始化虚拟机分配列表；$ms \leftarrow 0$；$TaskList \leftarrow Ant.TaskOrder$；

2:WHILE 调度列表存在未调度任务时 DO

3: *TaskId* 赋值为 *TaskList* 的首个任务;

4: FOR 每台 VM m in ER_{TaskId} DO

5: 根据算法 3.5 ComputeEFT 计算任务在每台虚拟机 m 的最早完成时间;

6: ENDFOR

7: 将任务分配给完成时间最早的那台虚拟机 $C^{m'}{}_{TaskId} \mid m' \in PR_{TaskId}, VM_{TaskId} \leftarrow m'$；
$s_{TaskId} \leftarrow s^{m'}_{TaskId}$， $C_{TaskId} \leftarrow C^{m'}_{TaskId}$；*TaskList* 删除任务 *TaskId*;

8:ENDWHILE

9:RETURN ANT

在算法 3.3 中，第 1 行首先初始化虚拟机分配列表以及完工时间并且复制任务调度顺序列表；第 3 行是根据任务调度顺序列表中选出第一个任务赋值给*TaskId*；第 4-6 行是遍历任务*TaskId*的可得虚拟机ER_{TaskId}，并且将任务*TaskId*在每个可得虚拟机上的开始时间和完成时间记录下，其中第 5 行是根据算法 3.5 计算出任务在每台虚拟机的开始时间以及完成时间；第 7 行根据之前记录的完成时间将任务分配给最早完成的那台虚拟机，随后将任务*TaskId*删除；最后得出整个任务调度的ms。

算法 3.4：基于虚拟机分配信息素模型采用轮盘赌法生成虚拟机分配列表

Function: *GenerateVMList*(*Pheromone_R*)

1: $Ant = \{VM_1, \cdots, VM_N\} \leftarrow \{null, \cdots, null\}$； $LD = \{ld_1, \cdots, ld_M\} \leftarrow \{0, \cdots, 0\}$

2: $MaxLd \leftarrow 0.01$；$TaskList \leftarrow Ant.TaskOrder$；

3: WHILE 调度列表存在未调度任务时 DO

4: *TaskId* 赋值为 *TaskList* 的首个任务; *TaskList* 删除任务 *TaskId*;

5: $P_{sum} \leftarrow \sum_{m \in ER_{TaskId}} \rho_m \times (1 - ld_m / MaxLd)^3$, $P_m \leftarrow \rho_m \times (1 - ld_m / MaxLd)^3 / P_{sum}$, $P_m \leftarrow \sum_{m' \le m} P_{m'}$;

6: 产生一个随机数 $\lambda \in [0,1)$;

7: FOR 每个 m in ER_{TaskId} DO

8: IF $\lambda < P_m$ THEN

9: $VM_{TaskId} \leftarrow m$;

10: $ld_m += TaskId.length / m.pc$;

11: $MaxLd \leftarrow \max\{MaxLd, ld_m\}$;break;

12: ENDIF

13: ENDFOR

14:ENDWHILE

15:RETURN Ant

在算法 3.4 中，第 1 行是初始化蚂蚁个体的虚拟机分配列表以及虚拟机初始负载；第 2 行初始化最大负载 $MaxLd \leftarrow 0.01$；第 5 行计算 ER_m 中每台虚拟机被任务 n 选中的概率 P_m 以及累计概率 P_m，其中 ρ_m 由虚拟机分配信息素进行状态转移结合启发信息计算出，启发信息根据虚拟机负载计算得出；第 6-13 行采用轮盘赌法随机选中一台虚拟机分配给任务 n，其中第 10 行是计算虚拟机的负载；第 11 行是更新最大负载值；最后返回蚂蚁个体。

3.2.4 解码方式

现实问题通过编码的方式形式化成调度方案，在蚁群算法中，编码方式决定了该算法搜索范围是否完备，即理论上能否找到最优解，相比之下，解码方式将形式化的调度方案具体为适应度值，适应度值往往是模型的目标函数的求解。不同的解码方式对于算法效率性能有很大程度的影响。常见的解码方式有：基于非插入式任务尽早开始的正向调度解码（EFT）、基于插入式任务尽早开始的正向调度解码（HEFT）、基于非插入式任务尽早开始结合 OCT 值的解码策略（OEFT）以及基于插入式任务尽早开始结合 OCT 值的正向调度解码（OHEFT）。

本文设计了正向解码和反向解码两种解码方式来解码个体，分别是基于插入式任务尽早开始的正向解码方法和基于插入式任务尽晚开始的反向解码方法，其中都会使用到同一种计算策略，因此将其单独抽象出算法 3.5。正向解码方式顾名思义是指任务调度顺序依据父子任务时序关系，调度当前任务前提是该任务的父任务全部被调度完；反向解码方式与之相反，首先调度是在 DAG 图中子任务

被调度完的任务，解码的是反向拓扑排序图。任何有效的个体都可以通过正向和反向解码成实际工作流调度方案。R_ACO 采用的是两阶段进化策略，第一阶段采用了基于 HEFT 算法解码个体同时生成虚拟机列表（算法 3.3），有效提高解质量，第二阶段则采用基于插入模式任务尽早开始的正向解码方式解码个体，具体如算法 3.6 和算法 3.7。

算法 3.5：计算任务在虚拟机上的最早完成时间

Function: $ComputeEFT(task,VM)$

1: $TaskId \leftarrow task$; $ATL \leftarrow \{[0, Max]\}$;

2: $rt_{TaskId} \leftarrow 0$;

3: IF $PR_{TaskId}.Size = 0$ THEN

4: $rt_{TaskId} \leftarrow \max\{C_{n'} \mid n' \in PR_{TaskId}\}$;

5: ENDIF

6：根据公式（2-5）计算出 pt_{TaskId} ;

7: 遍历 ATL 中可用时间段 $[pre, post]$，直至满足 $pre + pt_{TaskId} \le post \wedge rt_{TaskId} \ge pre$

8: $s_{TaskId} \leftarrow \max\{pre, rt_{TaskId}\}$，$C_{TaskId} \leftarrow s_{TaskId} + pt_{TaskId}$;

在算法 3.5 中，第 1 行定义 *TaskId* 为输入任务，初始化 *ATL*；第 2 行将任务的就绪时间初始化为 0；第 3-5 行获得任务的就绪时间，当任务不存在父任务时，就绪时间为父任务最晚完成时间；第 6 行根据公式计算出任务执行时间；第 7 行在 *ATL* 寻找可用时间段以满足条件；第 8 行获得任务的开始时间以及完成时间。

算法 3.6：基于插入模式任务尽早开始的正向解码方式

Function: $ForwardDecode(\&Ant = \{task_1,\cdots,task_N,VM_1,\cdots,VM_N\})$

1: 初始化 $ms \leftarrow 0$; $TaskList \leftarrow Ant.TaskOrder$;

2: WHILE 调度列表存在未调度任务时 DO

3: *TaskId* 赋值为 *TaskList* 的首个任务; *TaskList* 删除任务 *TaskId*;

4: 根据算法 3.5 Compute*EFT* 计算任务 TaskId 在虚拟机 RscId 的完成时间;

5:ENDWHILE

6: $ms \leftarrow \max\{C_1,\cdots,C_N\}$

在算法 3.6 中，第 1 行是初始化完工时间为 0 并且复制任务调度顺序列表；第 2-5 行执行一个循环操作以完成所有任务调度，其中，第 3 行根据任务调度顺序列表和虚拟机分配列表取出任务 *TaskId* 以及处理任务的虚拟机 *RscId* 随后从任务调度顺序列表中删除该任务；第 4 行根据算法 3.5 计算出任务在每台虚拟机的开始时间以及完成时间；第 6 行最后计算个体的适应度值即工作流完工时间 ms 。

算法 3.7：基于插入模式任务尽晚开始的反向解码方式

Function: $BackwardDecode(\&Ant=\{task_1,\cdots,task_N,VM_1,\cdots,VM_N\})$

1: 初始化 $ms'\leftarrow 0$; $TaskList\leftarrow Ant.TaskOrder$;

2: WHILE 调度列表存在未调度任务时 DO

3: $TaskId$ 赋值为 $TaskList$ 的首个任务; $RscId\leftarrow VM_n$; $TaskList$ 删除任务 $TaskId$;

4: 根据算法 3.5 ComputeEFT 计算任务 TaskId 在虚拟机 RscId 的反向完成时间;

5: ENDWHILE

6: $ms'\leftarrow \max\{C'_1,\cdots,C'_N\}$

在算法 3.7 中，第 1 行是初始化反向完工时间为 0 并且复制任务调度顺序列表；第 2-5 行执行一个循环操作以完成所有任务调度，其中，第 3 行根据任务调度顺序列表和虚拟机分配列表取出任务 $TaskId$ 以及处理任务的虚拟机 $RscId$ 随后从任务调度顺序列表中删除该任务；第 4 行根据算法 3.5 计算出任务在每台虚拟机的反向开始时间以及反向完成时间；第 6 行最后计算个体的适应度值即工作流反向完工时间 ms 。

3.2.5 改进策略

蚁群算法迭代进化与上一代个体高度相关，为防止个体陷入局部最优，本文设计了两种改进策略，在邻域进行搜索以寻求更优解。对于同一个蚂蚁个体采用基于插入模式任务尽早开始的正向解码和基于插入模式任务尽晚开始的反向解码可能会产生不同的完工时间，第一种改进策略是基于正反向循环解码个体改进策略（FBDNS）策略，持续对个体进行任务调度顺序变化直到无法再改进为止；第二种是基于负载均衡算法个体改进（LBNS）策略，LBNS 将分配给其它资源的任务重新分配给负载最小的资源来平衡资源负载，同时在选择要重新分配的任务时尽可能避免任务之间传输文件，即优先考虑已分配给负载最少的资源的任务的父任务或子任务，以缩短工作流的完成时间。基于正反向循环解码个体改进策略如算法 3.8 所示，基于负载均衡算法的个体改进策略如算法 3.9 所示。

算法 3.8：基于正反向循环解码个体改进策略(FBDNS)

Function: $FBDNS(\&Ant=\{task_1,\cdots,task_N,VM_1,\cdots,VM_N\})$

1: 定义一个标记值 $flag$ 初始化为 false;新生成一个个体 $NewAnt$ 并将输入个体赋值给它;

2: WHILE TURE DO

3： 定义一个个体 $OldAnt = NewAnt$;
4: 设置一个集合 TLBC 并将添加所有任务;
5: IF *flag* 为 false THEN
6: 对 TLBC 进行排序按照 $C^{OldAnt}_{task_n}$;
7： 定义一个值 n=0;
8： WHILE TLBC 列表存在未排序任务时 DO
9: $task_n$ 赋值为 TLBC 的首个任务; TLBC 删除任务 $task_n$;
10: ENDWHILE
11: $BackwardDecode(NewAnt)$;
12: IF $NewAnt.Fitness \geq OldAnt.Fitness$ THEN
13： break;
14: ENDIF
15： ELSE
16: 对 TLBC 进行排序按照 C'_{task_n} ;
17: 定义一个值 n=0;
18: WHILE TLBC 列表存在未排序任务时 DO
19: $task_n$ 赋值为 TLBC 的首个任务; TLBC 删除任务 $task_n$;
20: ENDWHILE
21: $ForwardDecode(NewAnt)$;
22: IF $NewAnt.Fitness \geq OldAnt.Fitness$ THEN
23: break;
24: ENDIF
25: ENDIF
26: $flag \leftarrow !flag$;
27: ENDWHILE

算法 3.8 中，第 1 行首先初始化标记值 *flag* ，其含义为正反向解码标记符号，$flag = true$ 时表示进行正向解码，反之表示为进行反向解码，随后生成一个新个体令其等于原个体；第 4 行将所有任务添加进 *TLBC* 集合；第 6-14 行表示反向解码策略，当 $flag = false$ 时，根据任务最晚完成时间优先选取任务，随即在 *TLBC* 集合中移除该任务，然后对操作后的新蚂蚁个体进行反向解码，以获取每个任务反向完成时间以及整个工作流的反向完工时间，如果新蚂蚁个体适应度值优于旧蚂蚁个体，循环退出，否则循环将一直执行直到个体没有改进为止；第 16-24 行表示反向解码策略，当 $flag = true$ 时，根据任务最小完成时间优先选取任务，随即在 *TLBC* 集合中移除该任务，随后对操作后新的蚂蚁个体 *NewAnt* 进行正向解码，以获取每个任务完成时间以及整个工作流的完工时间，如果新蚂蚁个体适应度值优于旧的蚂蚁个体，循环退出，否则循环将一直执行直到个体没有改进为止；第 26 行

表示每执行一次循环， *flag* 变化一次。

算法 3.9：基于负载均衡算法的个体改进(LBNS)策略

Function: $LBNS(\&Ant=\{task_1,\cdots,task_I,VM_1,\cdots,VM_I\})$

1： $OldAnt \leftarrow Ant$; $Ld \leftarrow \{0,\cdots,0\}$;

2: $ld_m \leftarrow \sum_{vm_n=m} pt_n^m$, $TSK_m \leftarrow \{n \mid vm_n = m\}$ $m=1,\cdots,M$;

3: $m' \leftarrow find\ \min\{ld_m \mid m=1,\cdots,M\}$; $RscSet \leftarrow \{null,\cdots,null\}$

4: IF $ld_{m'}=0$ THEN

5: *RscSet* insert $ET_{m'}$;

6: ELSE

7: $RscSet \leftarrow RscSet \cup \{PR_{TSK^n_{m'}} \cup SC_{TSK^n_{m'}} \mid n \in TSK_{m'}\}$;

8: *RscSet* erase $\{TSK^n_{m'} \mid n \in TSK_{m'}\} \cup \{task_l \mid m' \notin ER_l\}$;

9: IF $RscSet.Size=0$ THEN

10: *RscSet* insert $ET_{m'}$

11: ENDIF

12: ENDIF

13: SORT *RscSet* DESCEND BY $ld_{\bar{m}} \mid \bar{m} \in VM_l, l \in RscSet$;

14: $VM_{RscSet.begin} \leftarrow m'$;

15: $ForwardDecode(Ant)$; $FBDNS(Ant)$;

16: IF $Ant.Fitness > OldAnt.Fitness$ THEN

17: $Ant \leftarrow OldAnt$;

18: ENDIF

算法 3.9 中，第 1 行生成一个蚂蚁个体令其等于原蚂蚁个体，并将记录负载的集合初始化为 0；第 2 行表示计算出每台虚拟机的负载，并生成二维编码；第 3 行先是找出负载最小的虚拟机，然后初始化 *RscSet* 集合；第 4-12 行对 *RscSet* 集合进行增删操作，其中，当负载最小的虚拟机 m' 的负载为 0 时，将 m' 所有可执行的任务都加入至集合 *RscSet* 中；第 7-8 行表示当负载最小的虚拟机 m' 的负载不为 0 时，*RscSet* 集合进行增加以及删除的任务，最后如果 *RscSet* 集合为空的话，将 m' 所有可执行的任务都加入至集合 *RscSet* 中；第 13 行表示找出 *RscSet* 集合负载最大的一个任务，将其分配给虚拟机 m'；第 15 行表示将操作后的蚂蚁个体进行正向解码以及 FBDNS 改进策略；最终，如果改进后的蚂蚁个体适应度值更优，则返回该个体，否则返回原个体。

3.2.6 整体框架及算法复杂度分析

蚁群算法核心是根据蚂蚁行迹上信息素浓度而选择路径，通常信息素浓度较高的路径被选择的概率大于信息素浓度低的路径，一段时间后，较短路径上面信息素浓度会显著高于较长路径，周而复始，最后找到最佳路径。本文研究的基于蚁群算法的云工作调度问题中，目的是寻找最优工作流调度方案。将当前任务和紧接着调度任务抽象为信息素，同时也将任务和其分配的虚拟机抽象成信息素，通过对信息素进行状态转移生成任务调度顺序和虚拟机分配列表。采用两种信息素更新方式以寻求蚂蚁个体的更优解和增加蚁群多样性，全局信息素更新是根据当代蚁群中精英群体进行更新，局部更新并未实质性更新信息素，而是在构造蚂蚁解时，采用临时信息素以确保当代蚁群的多样性。在 R_ACO 中设计了两阶段进化策略，第一阶段是通过 Pheromone_R 对进行状态转移生成任务调度顺序后采用启发式算法生成虚拟机分配列表，因启发式算法存在搜索空间不完备的特质，往往搜索不到最优解，第二阶段是依次对 Pheromone_S 和 Pheromone_R 进行状态转移生成个体，同时采用改进策略进行邻域搜索，进而搜寻更优解。

其整体算法框架如算法 3.10 所示：

算法 3.10：改进蚁群算法

Function: R_ACO(& $Ant=\{task_1,L,task_N,VM_1,L,VM_N\}$)

1:初始化信息素模型 $Pheromone\_S$ 和 $Pheromone\_R$；

2: $Ant_{Best}\leftarrow$ 基于 HEFT 和 PEFT 两种方法生成较优的个体; $CurAntColony\leftarrow\varnothing$，

3:WHILE 第一阶段终止条件没有满足 DO //处于第一阶段进化

4: WHILE $|NewAntColony|<A$ DO

5: $Ant\leftarrow GenerateTaskOrder(Pheromone\_S)$； //信息素状态转移生成个体任务调度顺序列

6: $DecodeForMsByHEFT(Ant)$； //生成个体虚拟机分配列表并解码个体

7: $UpdateLocalPheromone$； //局部更新信息素

8: $NewAntColony\leftarrow NewAntColony\cup\{Ant\}$；

9: ENDWHILE

10: 更新 $CurAntColony\leftarrow NewAntColony\cup CurAntColony$ 保留前 A 个蚂蚁个体;

11: 从 $CurAntColony$ 中找出完工时间排名前 $\lceil Ap_e\rceil$ 的蚂蚁形成精英蚁群 $AntColony_e$；

12: 根据 $AntColony_e$ 更新 $Pheromone\_S$ 和 $Pheromone\_R$； //全局更新信息素

13:ENDWHILE

14:WHILE 算法终止条件没有满足 DO //处于第二阶段进化

15: WHILE $|NewAntColony|<A$ DO

16: $Ant \leftarrow GenerateTaskOrder(Pheromone\_S)$; // 信息素状态转移生成任务排序

17: $Ant \leftarrow GenerateVMList(Pheromone\_R)$; // 信息素状态转移生成虚拟机分配列表

18: $ForwardDecode(Ant)$; //蚂蚁个体解码

19: $UpdateLocalPheromone$; //局部更新信息素

20: $NewAntColony \leftarrow NewAntColony \cup \{Ant\}$;

21: ENDWHILE;

22: FOR $NewAntColony$ 中适应度值排名前$\lceil Ap_i \rceil$的每个蚂蚁个体 Ant DO

23: $FBDNS(Ant)$; $LBNS(Ant)$; //个体改进策略

24: ENDFOR

25: 更新 $CurAntColony \leftarrow NewAntColony \cup CurAntColony$ 保留前 N 个蚂蚁个体;

26: 从 $CurAntColony$ 中找出完工时间排名前$\lceil Ap_e \rceil$的蚂蚁形成精英蚁群 $AntColony_e$;

27: 根据 $AntColony_e$ 更新 $Pheromone\_S$ 和 $Pheromone\_R$; //全局更新信息素

28:ENDWHILE

在 R_ACO 框架中，第 1 行是对信息素模型 Pheromone_R 和 Pheromone_S 进行初始化操作；第 2 行是对蚁群进行初始化，同时基于 HEFT 或 PEFT 生成一个蚂蚁个体加入到新蚁群中；第 3-13 行是算法第一段进化，其中，第 4-9 行是根据信息素 Pheromone_S 进行状态转移生成任务调度顺序后采用任务最早完成时间生成虚拟机分配列表形成个体方案，生成每个蚂蚁个体之后，会进行一次局部信息素更新，以促成当代蚁群多样性，第 9-11 行是根据蚁群适应度排序选取若干个精英个体形成精英蚁群，第 12 行是根据精英蚁群对信息素进行全局更新；第 14-28 行是算法第二段进化，其中，第 15-21 行是首先依次对信息素 Pheromone_S 进行状态转移生成任务调度顺序和对信息素 Pheromone_R 进行状态转移生成虚拟机分配列表生成蚂蚁个体，直到蚂蚁数量达到蚁群规模，采用任务最早完成时间启发式算法解码个体，随后进行一次局部信息素更新，第 22-24 行是对蚂蚁个体采用 FBDNS 以及 LBNS 改进策略进行邻域搜索；第 25-26 行是根据蚁群适应度排序选取若干个精英个体形成精英蚁群，第 27 行是根据精英蚁群群对信息素进行全局更新，最后达到第二阶段终止条件后返回最优个体。

元启发式算法运行时间主要花费在迭代进化当中，R_ACO 也不例外，其算法时间复杂度分析如下：GenerateTaskOrder(Pheromone_S)的时间复杂度为$O(N^2)$，DecodeForMsByHEFT(Ant)的时间复杂度为$O(MN^2)$，UpdateLocalPheromone 的时间复杂度为$O(MN+N)$，GenerateVMList(Pheromone_R)的时间复杂度为$O(MN^2)$，

ForwardDecode(Ant)的时间复杂度为$O(N^2)$，FBDNS(Ant)的时间复杂度为$O(N^2)$，LBNS(Ant)的时间复杂度为$O(N^2)$。

更新 CurAntColony 的时间复杂度为$O(A^2)$，找出精英种群全局更新信息素的时间复杂度为$O(A^2p_e + Ap_eN^2 + Ap_eMN) \approx O(A^2p_e + Ap_eN^2)$，整个算法的复杂度为：

$$K_1\{A[O(N^2)+O(MN^2)]+O(A^2)+O(A^2p_e+Ap_eN^2)+O(NM+N)\}+$$
$$K_2\{A[O(N^2)+O(MN)+O(N^2)]+2Ap_iO(N^2)+O(A^2)+O(A^2p_e+Ap_eN^2)+O(NM+N)\}$$
$$=K_1[O(AMN^2)+O(A^2)]+K_2[O(AN^2)+O(A^2)]$$

其中，K_1，K_2分别为第一阶段和第二阶段进化的代数。

3.3 本章小结

本章节主要讨论基于两阶段蚁群算法下云工作流调度问题，首先介绍了蚁群算法原理并给出其应用场景，随后详细地阐述了提出的 R_ACO 算法细则，包括个体编码以及解码、信息素模型构造以及更新机制、状态转移概率规则以及构造个体解、改进策略。其中，在构造个体解部分设计了正反向两种解码方式，并给出了算法伪代码；在改进个体部分，设计了两种改进策略进行邻域搜索。最后给出了 R_ACO 算法整体框架并分析了其时间复杂度。

第4章 实验评价

本文提出的R_ACO算法研究的是在虚拟机资源确定情况下对工作流任务调度进行优化，以达到最大完工时间最小化。为了验证R_ACO算法的有效性，选取了以下算法进行对比分析，包括经典启发式算法 HEFT[12]、改进启发式算法PEFT[24]、元启发式算法GALCS[40]、混合算法HGA[43]、粒子群优化算法HPSO[47]以及分布式算法APRE_EDA[30]。

4.1 实验设置

4.1.1 案例选择

为公平、客观地验证本文提出的R_ACO算法的有效性，实验所选用的所有案例都源自于第三方案例库，确保本文算法和对比算法采用相同模型、相同案例下进行对比分析。实验案例可在案例库中自由下载，

https://confluence.pegasus.isi.edu/display/pegasus/Deprecated+Workflow+Generator

学者普遍认可这些案例，不同的工作流各具其特色，一定程度上满足研究需求。本文选用了三种不同案例类型的科学工作流模型，分别为 Epigenomics、LIGO、Montage.这些工作流案例来自不同的领域，具有不同的结构、通信和计算特点。图4.1为以上三种案例的结构布局。

以上三种科学工作流模型都源自于现实生活中，其中Epigenomics模型由南加利福尼亚大学基因中心Pegasus团队创建，其目的是能自动化处理不同的基因组序列；LIGO模型主要应用在物理学领域，用于生成和分析双星系统引力波状态；Montage模型由NASA/IPAC提出，主要应用于天文学界，输入多个图像创造出一种定制天空马赛克现象。

现实生活中工作流规模并非一成不变，而不同规模工作流采用的调度算法也不尽相同。为了验证算法的有效性，且具有普遍性，本文选用三种不同规模案例进行实验分析，其分别为小规模（约为30个任务节点）、中规模（约为50个任务节点）、大规模（约为100个任务节点）；考虑到现实任务调度过程中，资源具有特异性，即每种资源可处理的任务有限，由于本文探讨的是确定资源下工作

流调度问题，本文将资源可得率划分为三种，分别是 0.4、0.7 和 1.0,其中 0.4 代表该资源仅能处理任务总数×0.4 左右的任务，以此类推，1.0 表示该资源能处理工作流所有任务。

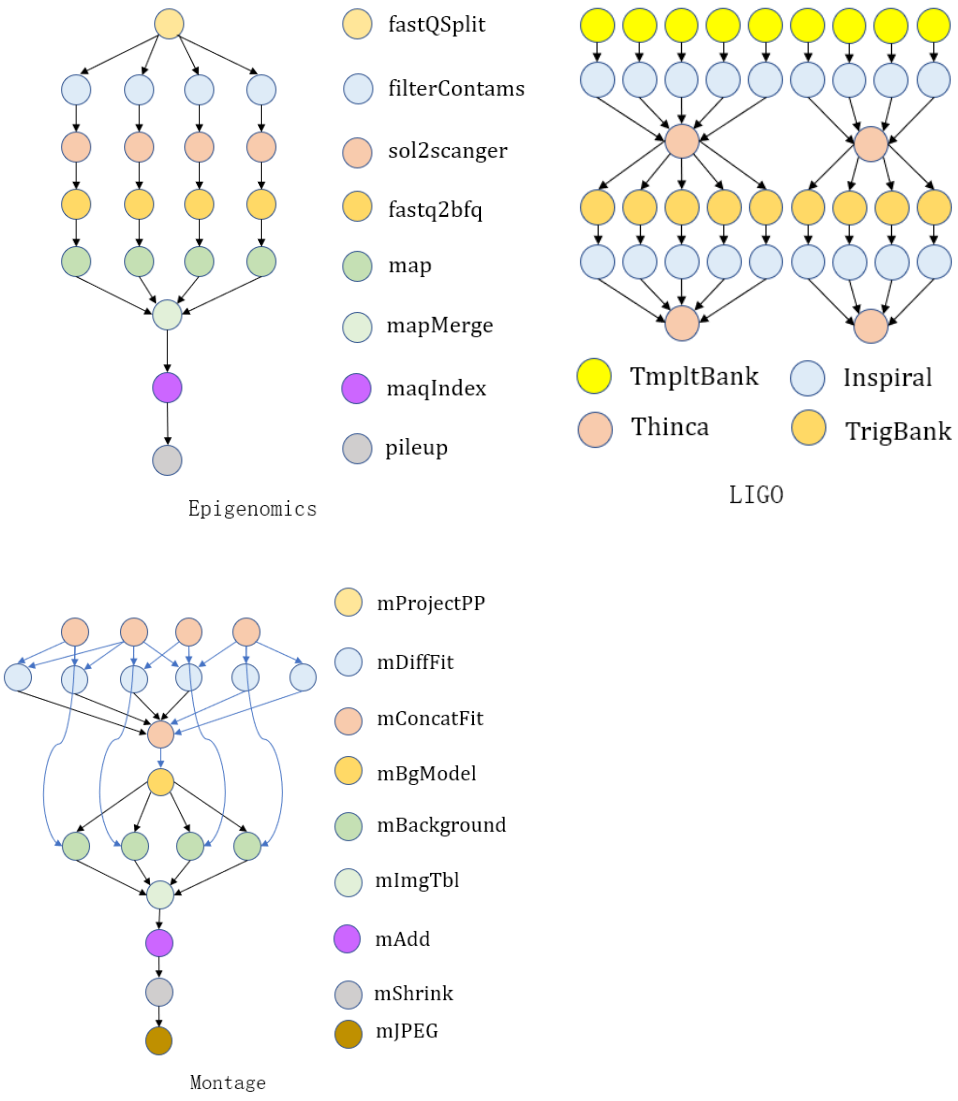

图 4.1 科学工作流案例 DAG 图

定义 $cs^{ty,sz,ra}$ 表示为案例类型，其中，工作流类型 ty 为 Epigenomics、LIGO、Montage，表示为 $ty \in TY = \{E,L,M\}$；工作流规模 sz 为 30、50、100，表示为 $sz \in SZ = \{S,M,L\}$；资源可得率 $ra = \{0.4,0.7,1.0\}$，以上案例对比可以较为全面验证 R_ACO 算法的有效性。

4.1.2 虚拟机配置

本文研究的是确定资源下云工作流调度优化问题，由于任务可并行处理，云工作流环境下可同时用不同类型的资源协同处理，具有异构性。本文设定了 6 台编号不同的虚拟机，将其运行在同一台计算机上，计算机具备 6 核及以上处理器，每台虚拟机的处理能力和带宽如表 4.1 所示：

表 4.1 虚拟机配置

虚拟机编号	r_1	r_2	r_3	r_4	r_5	r_6
带宽 (MB): bw_m	30	30	40	40	50	50
处理能力 (SCS): es_m	1.2	1.2	2.4	2.4	3.6	3.6

科学工作流案例中任务长度并不代表真实长度，是通过基准调整的，同时虚拟机处理能力和带宽已经进行相应比例处理。处理能力和任务长度比例应合理化，否则会形成较大误差。虚拟机资源可得率在上一小节已经详细说明，可得率定为 0.4、0.7、1.0 也是充分考虑到现实生活中的工作流情况。本文对比案例选取了 3 个，案例规模分别有 3 种，资源可得率也配置了 3 种，因此，实验采用案例共有 3*3*3=27 个，以验证算法有效性。

4.1.3 运行时间设置

本文中实验的程序代码以及对比算法程序是用 C++语言实现的，计算机运行环境为 LINUX 系统，其配 CPU 置频率为 3.20GHz，运行内存为 3.8G。由于对比算法的时间复杂度各不相同，为确保算法对比实验相对公平、客观，本文通过在相同的运行时间下对比不同算法适应度值。为使对比算法都尽可能收敛，本次实验设定算法收敛条件，对比算法达到算法收敛条件的时间可做为运行时间的参考，保证对比算法都充分搜索解。

HEFT[12]以及 PEFT[24]是经典的启发式算法，其算法运行时间和解方案都是根据案例确定的，不会自身变化，因此在运行时间实验中不做考虑该算法。而 GALCS[40]、HGA[43]、HPSO[47]、APRE_EDA[30]属于元启发式算法或混合算法，算法都会进行迭代进化以搜索更优解。但是，这些算法的时间复杂度都有差异。除开时间复杂度以外，这些算法的种群规模也不同，在资源可得率变动情况下，算法收敛所需时间也是动态变化的。当资源可得率较高时，任务分配虚拟机情况

更加复杂，从而搜索较优解时间大大增加。为充分考虑公平客观，本次实验设置每个算法收敛终止条件为连续迭代$200\sum_{n=1}^{N}|ER_n|/\sqrt[3]{N}/A$代没有改进，因算法存在随机因素，本次实验中选取的 27 种案例都会在 GALCS、HGA、HPSO、APRE_EDA 算法上运行十次以求得平均值，为保证所有对比算法都能充分收敛，选取四种算法中十次求平均运行时间最长的作为本次实验最终的运行时间，全部数据如表 4.2 所示。

表 4.2 运行时间

案例类型	规模	配置率	GALCS	HGA	HPSO	APRE_ED	Max
Montage	30	0.4	0.257	0.353	0.322	0.421	0.421
Montage	30	0.7	0.409	0.396	0.568	0.598	0.598
Montage	30	1.0	0.596	0.719	0.789	0.751	0.789
Montage	50	0.4	0.518	1.525	0.661	1.709	1.709
Montage	50	0.7	1.525	0.931	1.202	2.041	2.041
Montage	50	1.0	2.688	0.655	1.339	2.836	2.836
Montage	100	0.4	1.318	2.289	2.179	7.389	7.389
Montage	100	0.7	6.452	3.953	3.925	10.590	10.590
Montage	100	1.0	9.206	1.079	4.541	13.025	13.025
Epigenomics	24	0.4	0.125	0.258	0.208	0.318	0.318
Epigenomics	24	0.7	0.599	0.409	0.386	0.405	0.599
Epigenomics	24	1.0	0.580	0.614	0.463	0.821	0.821
Epigenomics	47	0.4	1.725	0.837	0.861	1.883	1.883
Epigenomics	47	0.7	2.937	1.409	1.103	1.331	2.937
Epigenomics	47	1.0	4.464	1.991	1.706	2.194	4.464
Epigenomics	100	0.4	1.124	0.783	1.768	4.082	4.082
Epigenomics	100	0.7	5.495	4.255	4.705	4.684	5.495
Epigenomics	100	1.0	3.705	3.288	4.885	10.524	10.524
Ligo	30	0.4	0.438	0.323	0.252	0.520	0.520
Ligo	30	0.7	0.964	0.996	0.461	0.699	0.996
Ligo	30	1.0	1.573	1.194	0.640	0.928	1.573
Ligo	50	0.4	1.636	1.199	0.695	1.223	1.636
Ligo	50	0.7	1.893	2.240	1.166	1.764	2.240
Ligo	50	1.0	3.085	1.362	1.470	2.163	3.085
Ligo	100	0.4	0.715	2.164	1.690	6.757	6.757
Ligo	100	0.7	6.541	4.554	4.195	9.311	9.311
Ligo	100	1.0	12.905	6.607	3.738	12.042	12.905

因此，对比实验运行时间如表 4.3 所示：

表 4.3 运行时间 $rt^{ty,sz,ra}$

Case	$cs^{M,S,0.4}$	$cs^{M,S,0.7}$	$cs^{M,S,1.0}$	$cs^{M,M,0.4}$	$cs^{M,M,0.7}$	$cs^{M,M,1.0}$	$cs^{M,L,0.4}$	$cs^{M,L,0.7}$	$cs^{M,L,1.0}$
$rt^{ty,sa,ra}$	0.421	0.598	0.789	1.709	2.041	2.836	7.389	10.590	13.025
Case	$cs^{E,S,0.4}$	$cs^{E,S,0.7}$	$cs^{E,S,1.0}$	$cs^{E,M,0.4}$	$cs^{E,M,0.7}$	$cs^{E,M,1.0}$	$cs^{E,L,0.4}$	$cs^{E,L,0.7}$	$cs^{E,L,1.0}$
$rt^{ty,sa,ra}$	0.318	0.599	0.821	1.883	2.937	4.464	4.082	5.495	10.524
Case	$cs^{L,S,0.4}$	$cs^{L,S,0.7}$	$cs^{L,S,1.0}$	$cs^{L,M,0.4}$	$cs^{L,M,0.7}$	$cs^{L,M,1.0}$	$cs^{L,L,0.4}$	$cs^{L,L,0.7}$	$cs^{L,L,1.0}$
$rt^{ty,sa,ra}$	0.520	0.996	1.573	1.636	2.240	3.085	6.757	9.311	12.905

4.1.4 参数设置

在 R_ACO 算法中，蚁群规模的选取极大影响算法时间复杂度，进而影响搜索效率及收敛时间，同时，信息素 Pheromone_S 以及 Pheromone_R 进行更新时，对应的挥发系数 ρ_1 和 ρ_2 值选择对算法性能也有显著影响。由此可见，参数选择对于算法性能至关重要。在 R_ACO 中，共设置了 6 个参数，分别为蚁群规模系数 β、信息素挥发系数 ρ_1 和 ρ_2、启发式信息权重 γ、改进率 p_i 以及第一阶段运行时间比例系数 θ。为确保选择出的参数组合有效，本文通过正交实验来选取实验参数。正交实验设计在很多领域研究中已经得到广泛应用，是研究多因素多水平的一种设计方法。以本实验 6 个参数为例，每个参数假定 5 个取值，倘若进行全面试验会生成一共 5*5*5*5*5*5=15625 组参数，这工作量无疑是巨大的。正交试验设计根据正交性从全面试验中挑选出部分有代表性的分布点进行试验，这种分布符合“均匀分散，齐整可比”特征，因此正交实验是一种高效率、快速、经济的实验设计方法。将 R_ACO 中 6 个参数填进正交表中可获取实验参数组合，6 个参数选取层次值如表 4.4 所示。

表 4.4 参数的各层次值

参数	层次				
	1	2	3	4	5
蚁群规模系数 β	1.0	1.5	2.0	2.5	3.0
信息素挥发因子 ρ_1	0.1	0.15	0.2	0.25	0.3
信息素挥发因子 ρ_2	0.1	0.15	0.2	0.25	0.3
启发式信息权重 γ	1.0	1.5	2.0	2.5	3.0
改进率 p_i	0.1	0.2	0.3	0.4	0.5
第一阶段终止系数 θ	0.5	0.55	0.6	0.65	0.7

其中，在 R_ACO 算法中，当代蚁群规模设置为 $A=\beta\times N$，β 为蚁群规模系数，

N 为科学工作流案例的任务个数；ρ_1 和 ρ_2 分别为任务调度顺序信息素模型 Pheromone_S 和虚拟机配置信息素模型 Pheromone_R 在更新信息素时的挥发因子；在任务调度顺序信息素进行状态转移时，通常会加入启发式信息，启发式信息权重对状态转移概率至关重要；在第二阶段蚂蚁个体生成后，会通过改进率 p_i 来对部分个体进行 FBDNS 以及 LBNS 改进；θ 表示为第一阶段运行时间占总运行时间比例，第二阶段运行时间则为剩余时间。随后根据正交表 $L_{25}(5^6)$ 进行参数组合，把实验次数减少至 25 次，每次实验都将案例类型、案例规模以及资源可得率考虑在内，同一组参数进行不同案例、不同规模、不同资源可得率实验，并对比分析。为了验证 R_ACO 算法鲁棒性，针对实验使用的所有案例都使用相同一组参数进行实验，并定义了平均响应变量以分析实验结果。其中，ARV_i 表示对于实验的第 i 组参数的平均响应变量，计算方式如下：

$$ARV_i = \frac{1}{tm \times |TY| \times |SZ| \times |RA|} \sum_{j=1}^{tm} \sum_{ty \in TY} \sum_{sz \in SZ} \sum_{ra \in RA} \frac{ms_{i,j}^{ty,sz,ra} - ms^{ty,sz,ra}}{ms^{ty,sz,ra}}$$

其中，tm 为每个案例每组参数运行的次数，$ms_{i,j}^{ty,sz,ra}$ 为算法在第 i 组参数下第 j 次实验中获得的工作流案例 $cs^{ty,sz,ra}$ 的完工时间，$ms^{ty,sz,ra} = \min_{j=1,\cdots,tm;i=1,\cdots,25} \{ms_{i,j}^{ty,sz,ra}\}$。

在正交实验中设置 tm 为 10，运行时间为 $rt^{ty,sz,ra}$，正交实验参数排列组合及实验获得的平均响应变量 ARV_i 如表 4.5 所示：

表 4.5 正交排列及获得的平均响应变量

run	Factors						
	β	ρ_1	ρ_2	γ	p_i	θ	ARV_i
1	1	1	1	1	1	1	0.01495945
2	1	2	3	4	5	2	0.01420104
3	1	3	5	2	4	3	0.01558838
4	1	4	2	5	3	4	0.01403924
5	1	5	4	3	2	5	0.01408174
6	2	1	5	4	3	5	0.01346475
7	2	2	2	2	2	1	0.01327950
8	2	3	4	5	1	2	0.01406346
9	2	4	1	3	5	3	0.01234630
10	2	5	3	1	4	4	0.01411392
11	3	1	4	2	5	4	0.01343611
12	3	2	1	5	4	5	0.01431035

13	3	3	3	3	3	1	0.01285423
14	3	4	5	1	2	2	0.01401200
15	3	5	2	4	1	3	0.01273836
16	4	1	3	5	2	3	0.01240983
17	4	2	5	3	1	4	0.01358624
18	4	3	2	1	5	5	0.01328081
19	4	4	4	4	4	1	0.01252850
20	4	5	1	2	3	2	0.01267722
21	5	1	2	3	4	2	0.01220773
22	5	2	4	1	3	3	0.01395066
23	5	3	1	4	2	4	0.01261340
24	5	4	3	2	1	5	0.01334647
25	5	5	5	5	5	1	0.01360913

表 4.6 各因素每个层次的平均响应变量及极差

Level	β	ρ_1	ρ_2	γ	p_i	θ
1	0.01457397	0.01329557	0.01338134	0.01406337	0.01373879	0.01344616
2	0.01345358	0.01386556	**0.01310913**	0.01366554	**0.01327930**	0.01343229
3	0.01347021	0.01368006	0.01338510	**0.01301524**	0.01339722	**0.01340671**
4	**0.01289652**	**0.01325450**	0.01361209	0.01310921	0.01374978	0.01355778
5	0.01314548	0.01344407	0.01405210	0.01368640	0.01337468	0.01369682
极差	0.00167745	0.00061106	0.00094297	0.00104812	0.00047048	0.00029012
影响程度排序	1	4	3	2	5	6
最优参数值	2.5	0.25	0.15	2.0	0.2	0.6

25 组实验进行计算分析得出每组参数对应的 ARV_i 值，表 4.6 根据计算出的平均响应变量 ARV_i 分析得出 R_ACO 算法最优的参数组合，从表中可以看到，本次正交实验中最终选出的参数为：蚁群规模系数 β 为 2.5，信息素挥发因子 ρ_1, ρ_2 分别为 0.25 和 0.15，启发式信息权重 γ 为 2.0，改进率 p_i 为 0.2，第一阶段终止系数 θ 为 0.6。通过每个参数的极差值可分析得出参数对算法影响程度，R_ACO 算法参数影响大小排序为：β、γ、ρ_2、ρ_1、p_i、θ。同时，所有参数任意层次的平均响应变量 ARV_i 都处于一个较低值水平，说明 R_ACO 算法在本次实验具有一定的鲁棒性。

4.2 结果对比和分析

为了验证算法效果，本章共进行两组实验，第一组实验共选取 3 种类型的科学工作流案例、3 种规模和 3 种虚拟机资源可得率共 27 个案例进行实验比较，分别和 6 种不同的算法 HEFT、PEFT、GALCS、HGA、HPSO、APRE_EDA 进行比较；第二组实验随机生成一个小规模案例，对比所有算法在相同代数下寻找最优解效率，并给出调度甘特图以直观描述对比结果。这些对比算法可代表启发式算法、元启发式算法以及两者混合算法。HEFT 算法以及 PEFT 算法是启发式算法，其每次运行结果是确定的，在对比实验中，只需要运行一次即可。GALCS、HGA、HPSO、APRE_EDA 这四种算法运行时会带有随机因素影响，每次结果会有小幅变化，为确保实验有效性，R_ACO 以及对比算法对 27 个案例分别运行 100 次得到 100 个数据，首先将数据统计出平均值进行对比分析，进而将所有数据制作成归一化箱线图并且计算出置信率，以验证 R_ACO 算法的有效性。

科学工作流案例本身源自于现实生活，不同案例之间数据量差距很大，由于本次实验选用的虚拟机资源处理能力及带宽是确定的，在案例不同规模任务数量情况下计算出的完工时间数值差距也很大，因此，本次实验对不同案例的完工时间做归一化处理。HEFT 作为启发式算法，在确定案例、规模以及资源可得率情况下，工作流调度方案以完工时间是确定的。本文将 HEFT 的完工时间作为基准，其他对比算法以及 R_ACO 算法工作流完工时间都除以 HEFT 完工时间。分别在图 4.2-4.4 描述了虚拟机资源可得率为 0.4、0.7 和 1.0 时，由这些算法获得的归一化工作流完工时间的平均值。

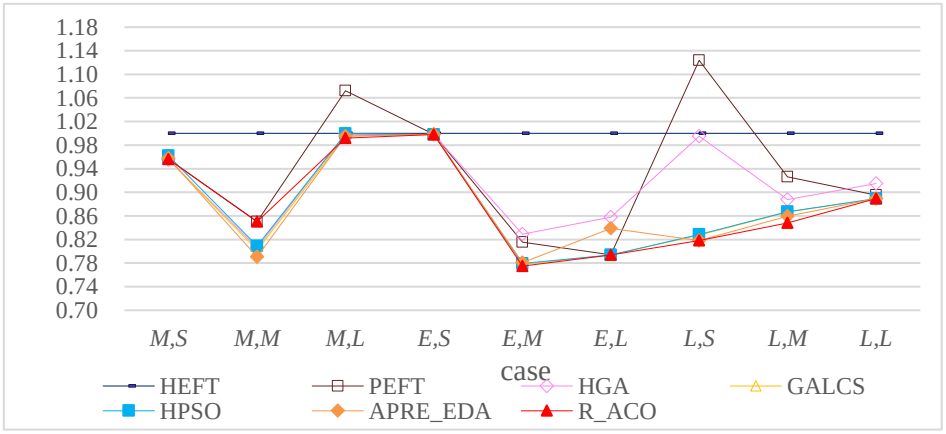

图 4.2 虚拟机资源可得率为 0.4 时各个算法获得的归一 makespan 平均值

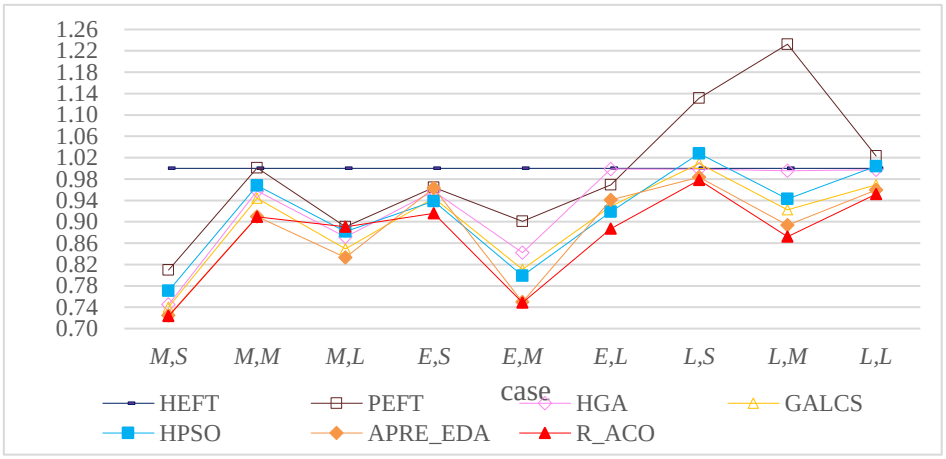

图 4.3 虚拟机资源可得率为 0.7 时各个算法获得的归一 makespan 平均值

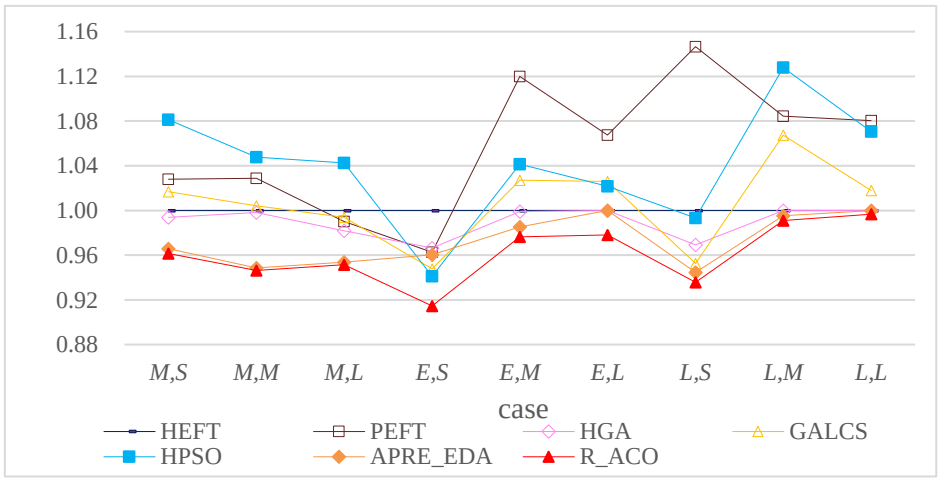

图 4.4 虚拟机资源可得率为 1.0 时各个算法获得的归一 makespan 平均值

由于本文优化目标是工作流调度完工时间，将不同算法在相同案例下获得的完工时间可得出算法优劣性。从图 4.2-4.4 中可以看出在虚拟机资源可得率为 0.4 时，在对比实验中所有元启发算法所有案例的结果都优于或等于 HEFT 的结果，PEFT 和 HEFT 在结果上相近；在虚拟机资源可得率为 0.7 时，HPSO 算法存在部分案例的平均完工时间超过了 HEFT 的完工时间；在虚拟机资源可得率为 1.0 时，HPSO 算法绝大部分案例平均完工时间都超过了 HEFT 的完工时间，GALCS 算法也有不少案例平均完工时间超过 HEFT 的完工时间，说明在虚拟机资源可得率较高的情况下，HPSO 和 GALCS 算法搜索容易陷入局部最优。而 R_ACO 在所有案例所有资源可得率中都没有比 HEFT 结果更差的情况，在绝大部分案例中，R_ACO 算法解质量都比其他算法更优。以上结果说明 R_ACO 算法是有效的，能够在所有解空间内全局搜索。进一步的，为评价 R_ACO 的具体改进程度，定义 R_ACO 相比于其它算法的平均改进率：

$$AMI_{***}=\frac{1}{|TY||SZ||RA|}\sum_{ty\in TY}\sum_{sz\in SZ}\sum_{sz\in RA}\frac{\overline{ms}_{***}^{ty,sz,ra}-\overline{ms}_{R\_ACO}^{ty,sz,ra}}{\max\left\{\overline{ms}_{***}^{ty,sz,ra},\overline{ms}_{R\_ACO}^{ty,sz,ra}\right\}}$$

其中： $\overline{ms}_{***}^{ty,sz.ra}$ 是由对比算法（HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA）获得的 $cs^{ty,sz,ra}$ 的平均工作流完工时间， $\overline{ms}_{R\_ACO}^{ty,sz,ra}$ 则是由 R_ACO 算法获得的 $cs^{ty,sz,ra}$ 的平均工作流完工时间。

在相同环境下运行 100 次获得的结果统计可得出，R_ACO 算法对于 HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA 算法平均改进率为 9.46%、8.46%、4.14%、2.38%、4.04%、0.70%，说明 R_ACO 算法实验效果是优于对比实验的其他算法。

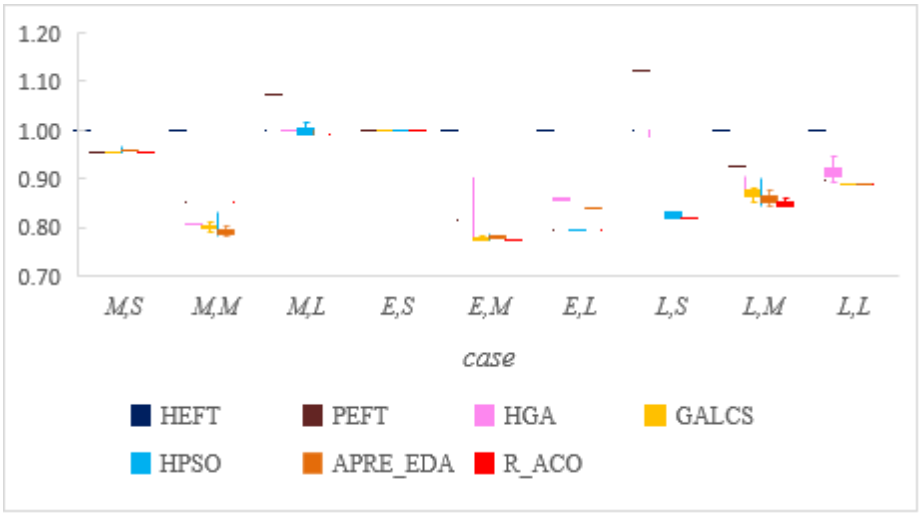

图 4.5 虚拟机资源可得率为 0.4 时各个算法获得的归一 makespan 箱线图

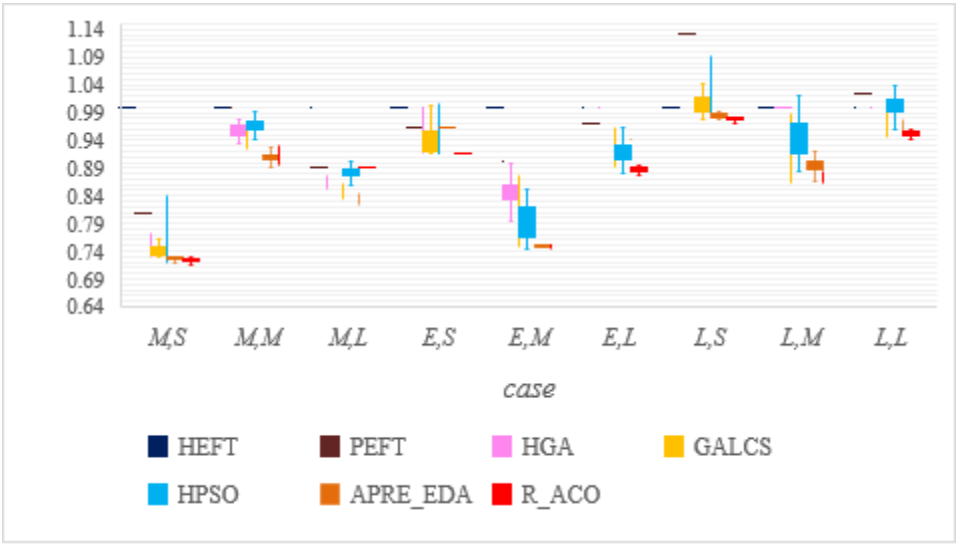

图 4.6 虚拟机资源可得率为 0.7 时各个算法获得的归一 makespan 箱线图

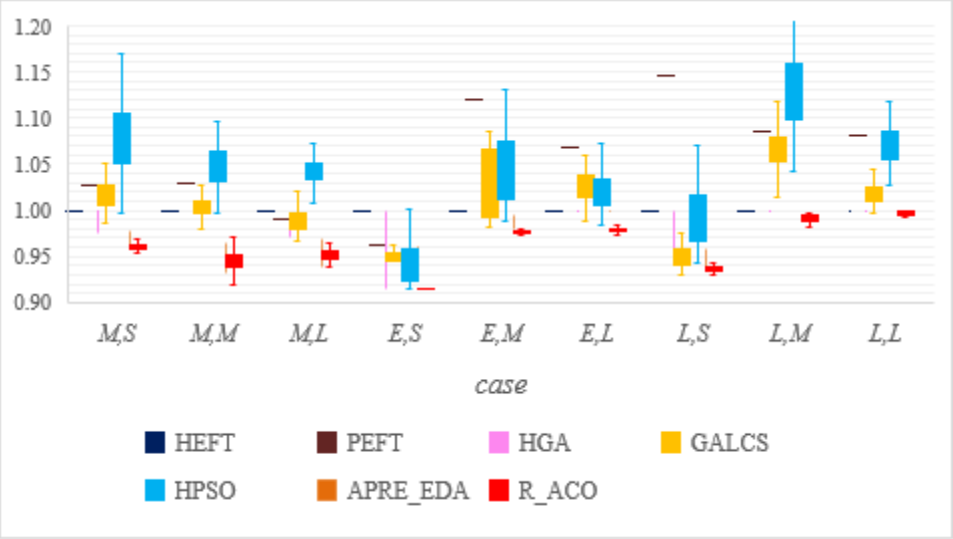

图 4.7 虚拟机资源可得率为 1.0 时各个算法获得的归一 makespan 箱线图

对比实验每种案例运行 100 次得到 100 组结果，将所有结果按照虚拟机资源可得率分类可得到箱线图 4.5-4.7。从图 4.5 可以看出，在虚拟机资源可得率为 0.4 时，绝大多数元启发式算法所有案例的 100 次运行所得完工时间值都小于或等于 HEFT 的完工时间；在虚拟机资源可得率为 0.7 时，HPSO、GALCS 算法存在部分案例 100 次运行中部分次数完工时间超过了 HEFT 的完工时间；在虚拟机资源可得率为 1.0 时，HPSO 算法绝大部分案例运行 100 次的完工时间都超过了 HEFT 的完工时间，GALCS 算法也存在大部分案例完工时间超过 HEFT 的完工时间。从图 4.7 中可以分析出，HPSO、GALCS 算法在运行 100 次结果中获得的完工时间较为不稳定，极差相对较大。说明 R_ACO 算法具有鲁棒性。

HEFT 和 PEFT 算法和其他 5 种元启发式算法不同，HEFT 和 PEFT 运行结果是依据案例确定的，而 GALCS、HGA、HPSO、APRE_EDA 算法和 R_ACO 算法在运行时会受到随机因素影响，从而导致每次得到的结果具有不稳定性，为进一步验证 R_ACO 算法有效性，将 R_ACO 与 HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA 算法 100 次运行结果两两比较，其计算方式为：

$$CP_{CA,ACO}=\frac{\sum_{i=1}^{I}x_{CA,ACO}}{I}\times 100\%$$

其中，$CP_{CA,ACO}$ 表示 R_ACO 算法与对比算法的置信率，$x_{CA,ACO}$ 表示决策变

量，当对比算法的 ms 值小于本文提出的 R_ACO 算法时，$x_{CA,ACO}=0$，否则 $x_{CA,ACO}=1$，I 为运行次数 100 次。针对这 100 次实验可计算出 R_ACO 算法在每个案例对于其他 6 种算法的置信率，通过表 4.7 可进一步分析出 R_ACO 算法优于 HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA 算法的平均置信率为：100.00%、100.00%、89.44%、83.74%、87.93%、82.15%。

表 4.7 R_ACO 优于（不差于）HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA 的置信率

案例	HEFT	PEFT	HGA	GALCS	HPSO	APRE_EDA
M,S,0.4	100.00%	100.00%	20.00%	24.00%	42.00%	100.00%
M,S,0.7	100.00%	100.00%	100.00%	100.00%	99.00%	66.00%
M,S,1.0	100.00%	100.00%	98.00%	100.00%	100.00%	70.00%
M,M,0.4	100.00%	100.00%	0.00%	0.00%	1.00%	0.00%
M,M,0.7	100.00%	100.00%	100.00%	100.00%	100.00%	60.00%
M,M,1.0	100.00%	100.00%	100.00%	100.00%	100.00%	56.00%
M,L,0.4	100.00%	100.00%	97.00%	64.00%	69.00%	98.00%
M,L,0.7	100.00%	100.00%	5.00%	0.00%	15.00%	0.00%
M,L,1.0	100.00%	100.00%	100.00%	100.00%	100.00%	61.00%
E,S,0.4	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
E,S,0.7	100.00%	100.00%	100.00%	96.00%	94.00%	100.00%
E,S,1.0	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
E,M,0.4	100.00%	100.00%	97.00%	92.00%	99.00%	100.00%
E,M,0.7	100.00%	100.00%	100.00%	99.00%	97.00%	81.00%
E,M,1.0	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
E,L,0.4	100.00%	100.00%	100.00%	31.00%	87.00%	100.00%
E,L,0.7	100.00%	100.00%	100.00%	100.00%	98.00%	100.00%
E,L,1.0	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
L,S,0.4	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
L,S,0.7	100.00%	100.00%	99.00%	100.00%	100.00%	95.00%
L,S,1.0	100.00%	100.00%	99.00%	88.00%	100.00%	93.00%
L,M,0.4	100.00%	100.00%	100.00%	95.00%	92.00%	85.00%
L,M,0.7	100.00%	100.00%	100.00%	95.00%	92.00%	85.00%
L,M,1.0	100.00%	100.00%	100.00%	95.00%	100.00%	97.00%
L,L,0.4	100.00%	100.00%	100.00%	87.00%	89.00%	95.00%
L,L,0.7	100.00%	100.00%	100.00%	95.00%	100.00%	83.00%
L,L,1.0	100.00%	100.00%	100.00%	100.00%	100.00%	93.00%
AVE	**100.00%**	**100.00%**	**89.44%**	**83.74%**	**87.93%**	**82.15%**

根据以上实验结果可知：R_ACO 算法是优于现在的一些经典算法（HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA），主要原因可归纳为：1）HEFT 以及 PEFT 都是启发式算法，其优点是短时间内可获得一种确定的工作流调度方案，

工作流调度方案高度依赖于案例，由于该算法无法也无需进行迭代进化，通常解质量不高； 2）HGA 是一种由启发式算法和遗传算法结合的算法，该算法基于启发式算法生成任务调度顺序，基于遗传算法生成资源分配列表，因其结合启发式算法，其任务调度顺序无法覆盖全部搜索范围，进一步导致整个算法无法进行全域搜索，存在找不到最优解的情况；3）GALCS、HPSO 和 APRE_EDA 三种算法都是元启发式算法。其中，GALCS 算法搜索空间是完备的，理论上可以找到最优解，但是算法步骤较为复杂，搜索效率不高。HPSO 是一种粒子群优化算法，其任务调度顺序和虚拟机分配都采用非整数编码方式，搜索空间是完备的，但是位置更新和速度变化是线性的，针对工作流调度优化其生成解具有一定随机性，因此搜索效率不高，容易陷入局部最优。APRE_EDA 算法是一种分布式估计算法，在任务调度顺序中引入概率模型，根据概率模型采样生成个体任务调度顺序，同时加入链路重构思想，将插入式调度方案转换为非插入式调度方案，但链路重构算法复杂度高，操作耗费时间多，导致算法搜索效率不高；4）R_ACO 算法的任务调度顺序编码和虚拟机分配编码都采用整数编码，保证搜索空间是具有完备性的，同时在任务调度顺序和虚拟机分配分别设计信息素模型，采用伪随机比例规则进行状态转移构造个体解，在信息素更新时采用局部更新和全局更新规则，同时采用两阶段迭代进化策略以增加算法搜索效率，也设计了个体改进策略增强邻域搜索能力，有效提高算法寻优能力和效率。

4.3 本章小结

本章进行了对比实验来验证 R_ACO 算法效果，由于云计算环境具有异构性，在选取虚拟机资源时，虚拟机执行能力是确定的，不同虚拟机可并发处理任务。选取了 3 种典型的科学工作流案例，并考虑了不同案例规模和不同资源可得率。通过设置收敛终止条件，确保对比算法都能收敛以确定实验运行时间。针对 R_ACO 参数选择，设计了正交实验，在给定运行时间下找到性能较好的一组参数。最后，在相同的运行环境下，对 R_ACO、HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA 算法进行了对比分析，从多个维度分析实验结果。实验表明，R_ACO 算法实验结果优于其他对比的算法，因此，本章实验验证了 R_ACO 算法在云工作流调度中的有效性。

第5章 总结与展望

5.1 总结

近年来，互联网技术快速发展，云计算环境下的工作流调度问题早已成为学界和工业界研究的热点问题，目前学者们已经有诸多研究成果。本文通过对比、分析现有云工作流调度优化算法，发现云工作流调度优化依然面临挑战，其中，部分优化算法在求解大规模工作流问题中略显乏力，通常需要更多时间且未必搜索到较优解；也有部分算法搜索解空间是不完备的，也就意味着存在找不到最优解的可能。本文在前人基础之上，对云工作流调度优化问题展开了深入研究，并提出了一种改进的蚁群算法（R_ACO）以求解云工作流调度问题。本文提出的R_ACO 算法在任务调度顺序和虚拟机分配都采用了整数编码，以确保搜索空间完备，并针对任务调度顺序和虚拟机分配建立了对应信息素模型，根据信息素进行状态转移构成个体解。同时，在整体框架上设计了两阶段进化策略，分为两阶段进行迭代进化，在第二阶段增加了个体改进策略，以增强邻域搜索能力。R_ACO 算法具备以下特点：1）采用局部更新和全局更新两种方式更新信息素，局部更新增加了当代蚁群多样性，全局更新提高了精英蚁群轨迹上的信息素浓度；2）状态转移概率规则中，启发式信息加入了虚拟机负载因素，以契合负载均衡思想；3）为增加算法邻域搜索能力，设计了两种改进策略，有效提高了算法搜索能力。在实验环节，本文首先设置了对比算法收敛条件，确保对比算法都收敛的情况下确定运行时间，随后采用正交实验获得最优参数组合，在相同行时间下对比 R_ACO 算法与 HEFT、PEFT、HGA、GALCS、HPSO 和 APRE_EDA 算法结果，经实验对比分析，R_ACO 算法效果优于对比算法（HEFT、PEFT、HGA、GALCS、HPSO、APRE_EDA），验证了本文提出的 R_ACO 算法在云工作流调度问题中的有效性。

5.2 展望

近几年来，从云工作流调度领域发展趋势来看，诸多学者研究该领域都以完工时间为优化目标，而在现实企业里，工作流完工时间固然重要，同时需要兼顾的因素也包括成本、可靠性、安全性。在全球生态环境每况愈下的今天，碳排放也是工作流调度值得考虑的一个问题。从实验结果来看，本文提出的改进的蚁群算法在优化工作流完工时间方面，对现有的优化算法有了较大提升，但仍有不足，未来将从以下几个方面进行研究：

（1）在优化目标方面，改进蚁群算法（R_ACO）虽然在优化工作流完工时间上比现有算法有显著提升，但 R_ACO 算法本质上是一种与问题无关的算法，具有良好的通用性和可扩展性。因此，R_ACO 算法可在评价指标增加上文提及的几个因素，可转换为多目标优化问题或者在一种约束下进行其他目标优化。

（2）在资源选用方面，本文研究的是在确定资源下进行工作流调度，即在调度开始前已经确定会使用几台虚拟机资源，以及虚拟机资源的处理能力已知。未来可在不确定资源情况下，研究云工作流调度问题，资源可根据调度需要随时租赁，按需付费。

（3）在工作流任务文件传输方面，本文使用一种数据共享中心来传输文件，这使得资源处理任务时，会把文件传输时间也一并计入资源执行时间，而另一种点对点模式（P2P）传输文件方式也值得进一步研究，P2P 模式核心是分布式思想，将虚拟机作为一种共享数据中心，任务文件可在虚拟机之间相互传输。

参考文献

[1] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4): 50-58.

[2] Zhan Z H, Liu X F, Gong Y J, et al. Cloud computing resource scheduling and a survey of its evolutionary approaches[J]. ACM Computing Surveys (CSUR), 2015, 47(4): 1-33.

[3] Mell P, Grance T. The NIST definition of cloud computing[J]. 2011.

[4] Surbiryala J, Rong C. Cloud computing: History and overview[C]//2019 IEEE Cloud Summit. IEEE, 2019: 1-7.

[5] 中国信息通信研究院. 云计算白皮书(2022 年)[M]. 2022 年.

[6] 柏昌顺,张新杰,王贵斌.仓储企业工作流管理系统研究[J].物流科技,2017,40(07):52-54+61.DOI:10.13714/j.cnki.1002-3100.2017.07.014.

[7] 李章兵,朱自兰.基于工作流的粮食筒仓储运信息系统设计与实现[J].计算机工程与应用,2002(06):227-230.

[8] Lu G, Tan W A, Sun Y, et al. QoS Constraint Based Workflow Scheduling for Cloud Computing Services[J]. J. Softw., 2014, 9(4): 926-930.

[9] 张鹏, 王桂玲, 徐学辉. 云计算环境下适于工作流的数据布局方法[J]. 计算机研究与发展, 2013, 50(3): 636-647.

[10] Jiang Y. A survey of task allocation and load balancing in distributed systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 27(2): 585-599.

[11] Kumar M, Sharma S C, Goel A, et al. A comprehensive survey for scheduling techniques in cloud computing. J Netw Comput Appl, 2019, 143: 1-33.

[12] Topcuoglu H, Hariri S, Wu M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE transactions on parallel and distributed systems, 2002, 13(3): 260-274.

[13] Zhang L, Zhou L, Salah A. Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments[J]. Information Sciences, 2020, 531: 31-46.

[14] Zhou X, Zhang G, Sun J, et al. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT[J]. Future Generation Computer Systems, 2019,

93: 278-289.

[15]Faragardi H R, Sedghpour M R S, Fazliahmadi S, et al. GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(6): 1239-1254.

[16]Deldari A, Naghibzadeh M, Abrishami S. CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud[J]. The journal of Supercomputing, 2017, 73(2): 756-781.

[17]Arabnejad V, Bubendorfer K, Ng B. Budget and deadline aware e-science workflow scheduling in clouds[J].IEEE Transactions on Parallel and Distributed systems, 2018, 30(1): 29-44.

[18]Bala R, Singh G. An improved heft algorithm using multi-criterian resource factors[J]. Int. J. Comput. Sci. Inf. Technol.(IJCSIT), 2014, 5(6): 6958-6963.

[19]Lin C, Lu S. Scheduling scientific workflows elastically for cloud computing[C]//2011 IEEE 4th International Conference on Cloud Computing. IEEE, 2011: 746-747.

[20]Rahman M, Hassan R, Ranjan R, et al. Adaptive workflow scheduling for dynamic grid and cloud computing environment[J]. Concurrency and Computation: Practice and Experience, 2013, 25(13): 1816-1842.

[21]Abrishami S, Naghibzadeh M, Epema D H J. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds[J]. Future generation computer systems, 2013, 29(1): 158-169.

[22]Bittencourt L F, Madeira E R M. HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds[J]. Journal of Internet Services and Applications, 2011, 2(3): 207-227.

[23]Sahni J, Vidyarthi D P. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans Cloud Comput, 2018, 6(1): 2-18.

[24]Arabnejad, H, Barbosa, et al. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table[J]. IEEE Transactions on Parallel and Distributed Systems: A Publication of the IEEE Computer Society, 2014, 25(3):682-694.

[25]Barrett E, Howley E, Duggan J. A learning architecture for scheduling workflow applications in the cloud[C]//2011 IEEE ninth European conference on web services. IEEE, 2011: 83-90.

[26]Zhou Y, Huang X. Scheduling workflow in cloud computing based on ant colony optimization

algorithm[C]//2013 Sixth International Conference On Business Intelligence And Financial Engineering. IEEE, 2013: 57-61.

[27]Zhu Z, Zhang G, Li M, et al. Evolutionary multi-objective workflow scheduling in cloud[J]. IEEE Transactions on parallel and distributed Systems, 2015, 27(5): 1344-1357.

[28]Wang Y, Zuo X. An effective cloud workflow scheduling approach combining PSO and idle time slot-aware rules[J]. IEEE/CAA Journal of Automatica Sinica, 2021, 8(5): 1079-1094.

[29]Wu C, Wang L. A multi-model estimation of distribution algorithm for energy efficient scheduling under cloud computing system[J]. Journal of parallel and distributed computing, 2018, 117: 63-72.

[30]Wu C, Wang L, Wang J. A path relinking enhanced estimation of distribution algorithm for direct acyclic graph task scheduling problem[J]. Knowledge-Based Systems, 2021, 228: 107255.

[31]Wang L, Siegel H J, Roychowdhury V P, et al. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach[J]. Journal of parallel and distributed computing, 1997, 47(1): 8-22.

[32]Meena J, Kumar M, Vardhan M. Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint[J]. IEEE Access, 2016, 4: 5065-5082.

[33]Rodriguez M A, Buyya R. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds[J]. IEEE transactions on cloud computing, 2014, 2(2): 222-235.

[34]Li Z, Ge J, Yang H, et al. A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds[J]. Future Generation Computer Systems, 2016, 65: 140-152.

[35]Akbari M, Rashidi H. A multi-objectives scheduling algorithm based on cuckoo optimization for task allocation problem at compile time in heterogeneous systems[J]. Expert Systems with Applications, 2016, 60: 234-248.

[36]Damodaran P, Vélez-Gallego M C. A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times[J]. Expert systems with Applications, 2012, 39(1): 1451-1458.

[37]Akbari M, Rashidi H, Alizadeh S H. An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems[J]. Engineering Applications of Artificial

Intelligence, 2017, 61: 35-46.

[38]Khajemohammadi H, Fanian A, Gulliver T A. Efficient workflow scheduling for grid computing using a leveled multi-objective genetic algorithm[J]. Journal of grid computing, 2014, 12(4): 637-663.

[39]Verma A, Kaushal S. Budget constrained priority based genetic algorithm for workflow scheduling in cloud[J]. 2013.

[40]Xia X , Qiu H , Xu X , et al. Multi-objective workflow scheduling based on genetic algorithm in cloud environment[J]. Information Sciences: An International Journal, 2022:606.

[41]Wu Q, Zhou M C, Zhu Q, et al. MOELS: Multiobjective evolutionary list scheduling for cloud workflows[J]. IEEE Transactions on Automation Science and Engineering, 2019, 17(1): 166-176.

[42]Wu Q, Ishikawa F, Zhu Q, et al. Deadline-constrained cost optimization approaches for workflow scheduling in clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(12): 3401-3412.

[43]Ahmad S G, Liew C S, Munir E U, et al. A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems[J]. Journal of Parallel and Distributed Computing, 2016, 87: 80-90.

[44]Keshanchi B, Souri A, Navimipour N J. An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing[J]. Journal of Systems and Software, 2017, 124: 1-21.

[45]Pandey S, Wu L, Guru S M, et al. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments[C]//2010 24th IEEE international conference on advanced information networking and applications. IEEE, 2010: 400-407.

[46]曹斌,王小统,熊丽荣,范菁.时间约束云工作流调度的粒子群搜索方法[J].计算机集成制造系统,2016,22(02):372-380.DOI:10.13196/j.cims.2016.02.010.

[47]Verma A, Kaushal S. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling[J]. Parallel Computing, 2017, 62: 1-19.

[48]Gonçalves J F. A hybrid biased random key genetic algorithm for a production and cutting problem[J]. IFAC-PapersOnLine, 2015, 48(3): 496-500.

[49]Wen Y, Xu H, Yang J. A heuristic-based hybrid genetic-variable neighborhood search algorithm

for task scheduling in heterogeneous multiprocessor system[J]. Information Sciences, 2011, 181(3): 567-581.

[50]Zhang M, Li H, Liu L, et al. An adaptive multi-objective evolutionary algorithm for constrained workflow scheduling in Clouds[J]. Distributed and Parallel Databases, 2018, 36(2): 339-368.

[51]Xu Y, Li K, He L, et al. A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems[J]. IEEE Transactions on parallel and distributed systems, 2014, 26(12): 3208-3222.

[52]Xu Y, Li K, Hu J, et al. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues[J]. Information Sciences, 2014, 270: 255-287.

[53]张建勋,古志民,郑超.云计算研究进展综述[J].计算机应用研究,2010,27(02):429-433.

[54]吕博文,杨怀洲.工作流技术综述[J].智能计算机与应用,2018,8(01):159-161.

[55]柴学智,曹健.面向云计算的工作流技术[J].小型微型计算机系统,2012,33(01):90-95.

[56]Vaquero L M, Rodero-Merino L, Caceres J, et al. A break in the clouds: towards a cloud definition[J]. ACM sigcomm computer communication review, 2008, 39(1): 50-55.

[57]罗干. 基于启发式算法的云工作流任务调度方法研究[D].北京交通大学,2015.

[58]Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 1996, 26(1): 29-41.

[59]Dorigo M, Maniezzo V, Colorni A. Positive feedback as a search strategy[J]. 1991.

[60]Gavish B, Graves S C. The travelling salesman problem and related problems[J]. 1978.

[61]Rodriguez M A, Buyya R. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments[J]. Concurrency and Computation: Practice and Experience, 2017, 29(8): e4041

[62]Xie, Yi, et al. "A two-stage multi-population genetic algorithm with heuristics for workflow scheduling in heterogeneous distributed computing environments." IEEE Transactions on Cloud Computing (2021).

致谢

时光匆匆如流水，转眼便是研究生生涯尾声，在此谨向帮助过的老师、同学们和家人们致以诚挚的谢意。

首先，我要感谢我的导师谢毅教授。谢老师博学多识、治学严谨、诲人不倦，是我学习生活的良师益友。他在我的学习生涯中，给予了莫大的帮助。在谢老师支持与指导下，本人顺利地完成了论文撰写以及科研实验。再次感谢谢毅老师对我成长路上教诲、支持和指导。

其次，我要感谢师兄们以及同学们。感谢桂奉献师兄以及汪煒军师兄对我学习上和生活上的帮助。感谢我的室友们和同学们，研究生生活因为有他们更加精彩，祝愿他们生活开心！

感谢浙江工商大学对我的培养。在浙商大的三年时光，遇到的老师与同学们使我受益匪浅！

最后，感谢我的父母我的姐姐以及我的女朋友小冰。在我的成长生涯中，家人给予了我无私的帮助，二十余年如一日地支持我。感谢小冰的陪伴！每当遇到挫折与烦恼，他们永远是我坚实的靠山与港湾，祝他们身体健康，万事胜意。

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含本人为获得浙江工商大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：　　导师签名：

签字日期：2022年 12月 8日　　签字日期：2022年 12月 8日

关于论文使用授权的说明

本学位论文作者完全了解浙江工商大学有关保留、使用学位论文的规定：浙江工商大学有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅，可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文，并且本人电子文档的内容和纸质论文的内容相一致。

本论文提交☑即日起/☐半年/☐一年以后，同意发布。

“内部”学位论文在解密后也遵守此规定。

学位论文作者签名：　　导师签名：

签字日期：2022年 12月 8日　　签字日期：2022年 12月 8日