**具有嵌套作业的装配调度问题优化研究**

摘要:

本文针对火箭焊接车间贮箱的加工装配过程提出了具有嵌套作业的装配调度问题模型，并基于logistic方程提出改进的遗传算法用以求解最小makespan和平均每个装配件的生产周期。装配贮箱之前需要将不同种类和数量的部件在不同机器上按照给定路线加工，由此将组件加工过程定义为嵌套操作以提取加工特征并进行数学建模。此外，模型中考虑了一定概率返工、动态加工时间等因素。为了改进GA，提出了更快的迭代方式来加速收敛，设计了柔性选择方式来保留较差解，设计新的交叉变异方式更新可行解，并设计自适应交叉变异概率来保留最优解。为了确定新算法的最佳参数，设计了正交试验，并在一些数值实验和经典问题(FT06, FT10 and FT20)中进行算法有效性和优越性验证。

关键词:复杂装配车间、调度问题、遗传算法、*logistic*模型、正交试验

1. Introduction

作为一种复杂的调度问题，Assembly Scheduling problem (ASP)广泛应用于制造业中,有较长的研究历史，装配生产系统主要用于生产不同种由多个组件装配而成的产品。每种产品由不同种类和不同数量的零件装配而成,而每种零件都有相应的一系列加工工序,不同工序在不同具有特定功能的机器上完成加工[1]。装配生产通常是人与机器的协同作业,同时对厂内物流系统也有较高的要求,总体可以分为装配作业车间调度问题(AJSSP)和装配流水车间调度问题(AFSSP),是批量生产企业的主要生产方式[2]。目前,装配车间普遍存在于生熟食品加工、饮料制造、制药、电子、半导体[3]、汽车、军工等各类大型制造业，除此之外,一些生产或处理过程也可以建模为装配调度问题,如分布式数据库的查询过程[4],发票制作过程[5]等.因此,对装配生产线的优化控制研究能为各类现实问题提供参考价值，具有现实意义。通常，大型复杂的装配件往往价格昂贵,如火箭、飞机、高铁等，每次优化都会大幅减少生产成本，而相关优化研究较少，因此对更加复杂的装配件调度问题进行研究有着更紧迫的需求。

装配调度问题包括加工阶段和装配阶段。其中，加工阶段比较接近普通的JSSP或FSSP，由此扩展研究为AJSSP[5]和AFSSP，装配阶段则类似单机调度。考虑装配作业的APS是比JSSP和FSSP更加复杂的调度问题，具有强*NP*-*hard*属性，对其进行求解优化有着较高的数学计算要求，因此研究此类问题在学术上也有较大意义。ASP存在较多扩展研究。AFSSP的第一阶段的加工机器存在两种布置情况: 一种是机器串行,即所有零部件需要按照一定序列按照依次转运到顺序机器完成置换或非置换流水加工; 另一种是机器或机器集并行,不同零部件在一些阶段可以并行独立加工。按照建模阶段数，进行了两阶段ASP和三阶段ASP的探索研究。近些年随着分布式计算的兴起，第一阶段被建模为并行且相同的工厂，每个工厂内是连续的串行机器，即分布式ASP。AJSSP则以JSSP开始，以装配作业结束，其加工灵活性较大，研究者针对此类问题提出了多种模型和求解算法，根据可用设备数量，可以建模为柔性AJSSP，且AJSSP常用于批量流（lot streaming）问题中。一般作业车间调度问题往往较难优化完工时间，具有装配作业的AJSSP尤其如此. 以上两者或者两者的衍生研究问题的共同点是,所有装配产品的组件加工流程相同,不同组件分别在不同的生产线进行独立加工,不考虑重加工或返工，而关于具有多种数量不同工序不同混合加工(即本文定义的嵌套加工)部件的装配产品尚未有详细研究。

传统装配调度问题如图1a所示，例子中不同装配件具有相同结构，由同类型数量为一的组件装配而成。每种组件经过给定的加工路线之后再经装配作业得到成品，图中给出了两个装配件的调度甘特图，其完工时间分别为*C*1和*C*2。本文考虑的复杂ASP如图1b所示，图中有多种装配件，每种装配件包含多种组件，组件之间数量之比为**. 不同装配件的比例不同，不同组件的部分加工阶段存在共用机器，组件与机器的分配是一项重要任务。某些组件经过加工后若检测不合格则需要返工，再次检测合格后方能进行之后的加工。图中显示了当**时的甘特图，由图可以看出四种组件的加工位置不完全相同，工序排序有多种方案，不存在返工工序时，装配件的完工时间为*C*1，所有工序时间计算一次；当个别组件产生返工时，最后的完工时间将会从返工工序开始再次累计计算，如*C*2中O41和O42计算两次，其他工序的时间一次. 当各组件数量更多时，其中过程将更加复杂。本文将此类不同组件轮番加工且工序数量不同的加工过程称为嵌套加工，并将其应用到火箭贮箱焊接车间的装配调度问题中，然后提出改进算法进行多层优化。

本文贡献如下：

（1）定义了嵌套加工工序，建立具有嵌套工序的ASP整数规划模型；

（2）以logistic方程为基础，对遗传算法（GA）设计多种改进措施；

（3）设计正交实验，用以确定改进算法的最佳参数；

（4）以提出算法进行随机案例试验，验证提出ASP模型和算法的有效性。



图1*a*. 一般装配调度问题



图1*b*. 具有嵌套作业的装配调度问题

2.Literature review

2.1 AFSSP及其拓展研究文献综述

在实际生产过程中,装配流水车间主要包含加工、运输和装配三个过程, 然而运输阶段在大多数情境中较为简单，常常被忽略，按照是否考虑装载运输过程将其分为两阶段AFSSP和三阶段AFSSP,不少学者根据问题类型建立了相关模型并提出不同方法对其展开研究。例如，*Vairaktarakis*和*Lee*介绍了一种具有加工灵活性的两机流水车间,其假设每个工件均需经过两个先后约束的工序,其加工模式为两台机器分别先后加工两个工序或两台机器协同加工两个工序, 为了确定每个任务的加工模式,同时为后续调度保留相应约束条件,采用了动态规划的方式求得了大中型问题的最优解,并提出一种多项式启发式算法进行对比试验. Zhao等人[22]将AFSSP看作一种具有装配约束的FSSP,考虑最小能耗的目标建立了关于实际转向器的节能调度模型,并采用混合差分进化算法进行了求解。为了积极应对竞争压力并提高实际生产能力,ASP越来越考虑现实环境,从而满足实际需要。*Zhang*和*Wang*考虑现实的动态制造环境集成柔性装配,建立具有非线性工艺和依赖于序列设置时间的*MILP*模型,设计带有机器反馈机制调度规则的约束规划进行求解。*Yavari*和*Isvandi*将包括零件供应商、零件制造商和装配商的三级供应链建模为两阶段装配集成调度问题,以零件订单数量和日期为决策变量来最小化总加权完工时间、零件订购和保持成本的总和,并以*GA*和*CPLEX*进行实例试验以对比两者的性能。

*AFSSP*的扩展研究多数考虑新的约束条件或工作场景，由此研究者提出了多种有价值的新模型。如，*Zhang*和*Xing*先后考虑有不同的设置时间的分布式柔性装配流水车间调度问题（DFAFSSP）,并设计算法进行实例研究,在数值上验证算法的有效性和效率。之后,*Hatami*等人提出两阶段分布式置换流水车间调度问题,其中第一阶段为若干相同的置换流水车间工厂或加工中心,第二阶段为装配阶段。*Chen*等人针对公司降低成品库存同时完成生产和准时装运的目的,建立关于最短平均最长等待时间和平均提前和拖期的两阶段*AFSSP*模型,其中第一阶段存在两种工序数不同的加工方式。然后提出改进的*GA*进行试验,为实际生产提供了参考意见。而且，Wu等人研究了集成的两阶段三机*AFS*、多智能体调度和与时间相关的加工时长问题模型,在确定一个智能体总完工时间上限基础上,最小化另一个智能体的总完工时间,以分支定界算法推导三个小规模问题下界,然后提出四种混合粒子群算法进行大型问题求解。*Yokoyama*提出了一种包括加工、安装和装配的*AFSSP*,针对由多个单操作项目（部件）组成的产品,每个产品的零部件第一阶段为单条流水线加工,机器之间存在设置时间。为了最小化所有产品的平均完工时间,将所有操作划分成块,然后采用伪动态规划和分支定界算法进行求解。

由于AFSSP是NP-hard问题，其求解算法也有较多研究。*Jung*等人考虑不同部件切换时的设置时间, 提出了三种GA进行求解具有动态组件尺寸产品的*AFSSP*。*Komaki*等人研究了三阶段AFSSP,为了求得最小*makespan*,提出改进的布谷鸟算法并进行验证。*Wang*和*Zheng*提出了改进的离散蝙蝠算法进行求解此类*AFSSP*。Pan和Gao等人针对两阶段分布式置换流水车间调度问题以*makespan*为目标设计启发式算法,并在810个基准实例上验证所提出算法的高效性。*Ali*和*Fawaz*将分布式数据库系统中的查询问题建模为AFSSP,以最小化所有查询延迟为目标,提出三种启发式算法:*EDD*、*PSO*、禁忌,通过实验证明*PSO*在紧到期日时表现最好,而禁忌在松弛到期时间时表现最好。*Li*等人研究了一种具有批量交付约束的分布式流水车间调度问题,生产阶段为置换流水车间调度过程,而将批量交付阶段建模为装配模型,提出了结合鲸鱼优化算法和局部搜索的混合算法,在实际应用案例上进行验证。*Li*和*Xing*针对柔性装配系统（*FAS*）的调度问题,考虑了易死锁情况,结合*FAS*的*Petri*-*net*模型,提出启发式波束搜索算法来求解最小*makespan*。之后,两人和*Lu*考虑没有缓冲区的*FAS*调度问题,增加了阻塞和死锁约束,对*FAS*建立*Petri*-*net*模型,提出混合粒子群算法来最小化*makespan*。*Zhao*等人针对现实可存在的分布式装配无空闲流水车间调度问题,提出*CWWO*的协作水波优化算法来最小化最大装配完成时间,设计了一种基于变邻域搜索算法（*VNS*）的强化学习机制在传播阶段平衡算法的探索和利用,引入增强局部搜索能力的破坏算子对算法进行设计,最后在基准集问题上验证了算法的高性能。Lu等人将订单审核/发布 (ORR)应用到AJSSP中，评估了ORR的降低库存成本和缩短订单流动时间的能力。

2.2 AJSSP及其扩展研究文献综述

Thurer等人研究了AJSSP中设置Due Dates (DDs)的问题，并进行工作负载控制（WLC）。考虑两级装配车间的两种情况:订单集中在一个总装操作上和两级多阶段作业车间进行一系列组装操作，提出了新的调度规则。Gomes等人考虑了按订单生产AJSSP中的重入过程，优化订单提前、延迟和在制品库存的加权和，并设计了针对性算法求解。Wong和Ngan研究了AJSSP的批量流(lot steaming)LS技术用于批次分割，考虑组件共享和系统拥堵指数，提出混合遗传算法（HGA）和混合粒子群优化（HPSO）来解决问题。Wan和Yan研究了知识制造的集成AJSSP与自重构的问题，以成本最低为目标，开发了启发式算法优化了最佳调度提高了工作站利用率。Borreguero-Sanchidrian等人研究了航空业的FAJSSP，提出优化操作员数量的机身数量，降低机身的生产成本。Liao和Wang优化了AJSSP的装配成本与碳排放的帕累托最优问题。随着对环境保护问题的重视，相关研究也在不断增加。考虑到资源的限制，Thurer等人研究了AJSSP具有双重资源约束的工人分配问题，模拟了工人分配的效益影响。

GA或混合GA和其他启发式算法的研究在JSSP中得到了大量的研究，进一步在AJSSP中得到了广泛的研究。针对晶体管-液晶显示器(TFT-LCD)的AJSSP，Chou等人以makespan、总加权迟到作业数、总设置时间最优为目标，提出了一种多目标混合GA，与 variable neighborhood descent (VND) algorithm混合进行局部搜索和评估，从而解决了所提问题。Surjandari等人针对具有生产多项目多级产品并行机器的AJSSP，研究了静态和动态条件下的到期日问题，优化了总流动时间。Wu等人研究了分布式the distributed assembly flexible job shop scheduling problem (DAFJSP)，以最小化提前迟到总成本为目标，设计了改进的差分进化模拟退火算法进行求解。Zhang等人针对动态柔性ASSP优化了三个子决策, 包括释放决策、路线决策和排序决策。 Daneshamooz等人针对具有并行装配和批次流的FAJSSP，将其建模为混合整数规划模型，基于VNS和自适应VNS设计算法求解最小化的makespan。近些年随着对环保问题的重视，相关研究逐渐增多。Ren等人考虑能源消耗和生产效率的柔性作业车间调度问题（FJSP）联合优化研究，提出集成PSO和GA的启发式算法优化了生产效率与能源消耗。以及新型机器学习应用，Wang等人针对装配车间生产环境的不确定性，结合强化学习的实时性特点，提出了一种双Q-learning（DQ）方法，通过自学习增强装配车间调度对环境变化的适应性。Wang和Lu针对离散制造的集成AJSSP，优化加工顺序和装配顺序，最小化makespan和总库存时间，并设计了非支配排序GA-II(NSGA-II)。Lin等人同样为了同时优化作业处理和装配，提出作业约束的FAJSSP，并提出相应的作业约束GA优化问题。

3.问题描述及数学模型

3.1 嵌套操作调度问题描述

本文研究的装配调度问题是基于火箭贮箱的焊接工艺，可以完成一些火箭贮箱的生产和装配任务，如图2所示。 在引入嵌套操作之前，有必要介绍贮箱的加工过程。每个储罐由三种数量不同的部件组成 (*h*1, *h*2 and *h*3) 并有不同的组件加工工艺。罐体产品的整个过程可以建模为两阶段装配调度问题：第一阶段是完成组件生产，第二阶段是装配所有组件。



图2. 火箭贮箱的焊接工艺



图3.三组件的加工过程.

在图3所示的所有机器中，串行机器M1和M2只能完成组件1和组件2的前两个操作。并行机M3和M4具有相同的功能，即各自可以独立地完成组件1、组件2或组件3的工序。 与M3和M4不同，并行机器M5和M6只适用于组件3。 至于其余机器，M7可以加工三种组件，而M8只加工组件3。 另外，需要注意的是，组件3在被M3或M4加工后，有一定的概率返回M6或M5或M6。 在M7或M8之后可以获得成品，然后在下一阶段用M9、M10和M11机器组装。

解决方案是找到一个没有优先级的所有储罐的优化顺序，可以根据组装过程划分为零件序列，并进一步分为序列嵌套串行和并行操作。如图4所示的问题有两层优化作为嵌套特性。 第一层也是母层，它是为了优化n个罐的数量顺序。根据图2和图3，每个储罐可由三种部件组装，总数为h1+h2+h3。因此，所有的罐可以分成不同顺序的组件作为第二个优化层。因此，所有的罐需要分成有不同加工顺序的组件优化问题，即第二优化层。 由于零件和储罐的种类和数量都不同，如果数量和种类不同是确定，优化的规模可能不确定，导致问题难以用算法进行编码。此外，被分配的机器不是固定的，加工时间是动态的。而且，如果操作不合格，则发生返工，造成额外工艺和机器占用。这个具有返工优化问题的多层问题可以被建模为具有嵌套操作的ASP。



图4. 调度问题的嵌套特性

根据以上描述，火箭罐焊接工艺的第一阶段比一般装配调度问题更复杂，有必要建立新的数学模型，提出新的算法。

3.2 数学模型

本文研究的火箭罐焊接车间包含Ɲ种贮箱和种设备。在装配前的机械加工阶段， let 表示一组处理机器，并行机器的数量 *mj* 为 , 所以各种机器的总数是; 让所有的机器都用整数 1到**来表示, 则 1 到**可以代表机器*m*1, to**可以代表机器 *m*2,…, 等等。同样， let *Ɲ* 表示所有火箭贮箱的集合, *ui* 是贮箱的数量 *qi*, 因此，该系统中的贮箱的总数是 *u* = *u*1+ *u*2 +…+ *u**Ɲ*. 同样，用数字1-u表示所有贮箱，则1~u1表示贮箱q1，u1+1~u2表示q2,…,依此类推。所有贮箱均由三种组件H1、H2和H3装配而成，不同贮箱的组件配套比例不同，贮箱qi的三种组件（H1、H2和H3）配套比例为*hi*1:*hi*2:*hi*3（未约至最简），贮箱qi的组件总数为*hi*=*hi*1+*hi*2+*hi*3。用数字1-hi表示组件，则1~ *hi1*表示组件1，*hi*1+1~ *hi*2表示组件2，*hi*2+1~ *hi*3表示组件3。

在该车间内，首先将未加工组件按照给定加工路线进行一系列工序加工，三种组件按照一定比例完成后进行组件配套，进入产品装配加工阶段，装配阶段为流水加工作业，其加工过程可以简化为一台机器的作业A，设其加工位置为M(A)，加工时间为o(A)，最终得到一个成品。组件Hν的工序数为*Lv*，设其加工路线为*ψv*=*σv*1*σv*2*…σvlσv*(*l+*1)*…σvLv*。 M(*σvl*)为工序*σvl*的加工机器，则给定加工路线*ψv*，其可以用有一定顺序的加工机器表示，即M(*ψv*)= M(*σv*1) M(*σv*2) …M(*σvLv*) 。*σvl*的加工时间用*ο*(*σvl*)表示， , 其中X为加工时间均值，X/10为标准差，[X-X/5,X+X/5]为截尾的边界。

组件的第一道工序*σv*1对应的加工机器M(*σv*1)存在一定概率加工不合格，经检测工序检测后返回该机器重新加工，返工概率用γ(*σv*1)表示。组件的前后工序之间装卸、搬运由工人和AGV完成，其作业时间用š(*σvlσv(l+1)*)表示，服从[x-x/10,x+x/10]的均匀分布 *U* (*x*-*x*/10, *x* + *x*/10), x为转运时间均值，其大小与加工时间X的比值在一个较小的范围。

其他说明如下：子装配被忽略；任何连续机器之间都缓冲区足够大；所有组件都是独立的，可在时间零点进行加工；在任何时候，每台机器最多可以加工一个零组件，并且每一个零件最多可以由一台机器加工；每个工序只需要一台预定的机器进行处理；没有抢占发生。优化目标是总完工时间最短以及三种贮箱的平均生产周期最小。

本文有两个目标需要优化。第一个是最小化所有贮箱开始和结束之间的时间差（makespan）。第二个是最小化每个贮箱开始和结束之间的时间差（生产周期）。

为了讨论，给出了以下定义:

(1) Indices

*i*: 贮箱类别的序列号, *i* =1,2, …, *Ɲ*;

*j*: 机器类别的序列号, *j*=1, 2, …, ;

*v*: 零件类别的序列号, *v*=1,2,3;

*a*: 贮箱的序列号, *a*=1, 2, …, *u*;

*b*: 机器的序列号, *b*=1, 2, …,**;

*k*: 组件序列号, *k* =1, 2, …, *hi*;

*l*: 操作的序列号, *l*=1, 2*,* …*,* *Lv*;

*A*: 装配操作的索引;

(2) 参数

: 机器类型的数量;

: 类*j*机器平行机的数量;

*Ɲ*: 储罐类型的数量;

*hiv*: 零件数量 *H*ν of tank *qi*;

*γ*(*σv*1): 操作中返工的概率 *σv*1;

*X* (*Oakl*):理论上*Oakl*平均加工时间;

*x* (*Oakl*, *Oa,k,l*+1): 平均经过时间 between *Oakl* and *Oak,l*+1 ;

: *x* (*Oakl*, *Oa,k,l*+1)与 *X* (*Oakl*)的比率;

*o*(*OaA*):装配操作的机加工时间 of the *ath* tank;

(3) 变量:

*qi*: 带有序列号的贮箱种类 of *i*;

*H*ν: 具有序列号的零件种类 of *v*;

*SaA*: 装配操作的开始时间 of the *ath* tank;

*CaA*: 装配操作的完成时间 of the *ath* tank;

*T*(*a*)= *qi*: the *ath* 贮箱被归类为*qi*;

*σvl*: the *lth* 组件操作 *Hv*;

*Oakl*: the *lth* 组件操作 *Hk* of the *ath* tank;

*OaA*: 装配操作 of the *ath* tank;

*τ*(*σvlσv*(*l+1*)): 相邻作业的转移时间;

*ο*(*σvl*): 加工时间 of *σvl*;

*Saklb*: 在b机上进行机械加工 *Oakl*开始时间;

*Caklb*:在b机上进行机械加工 *Oakl*结束时间;

*u*: 订单中的贮箱总数, *u*=*u*1+*u*2+…+ *ui*+…+*uƝ*.

*mj*:机器类被 *j*;

*M*(*Oakl*): 加工机of *Oakl*;

*M*(*OaA*): 加工机 of *OaA* 可以设置为模型中的最后一台机器;

*ρ*: 调整参数，可以决定起始时间的灵活性;

*sb*:机器*b*的设置时间;



*hi*: 贮箱的所有部件的编号 *qi*, *hi*=*hi*1+*hi*2+*hi*3;

**: 机器总数, ;

(3) Objective function:

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |

Subject to

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |
|  | (5) |
|  | (6) |
|  | (7) |
|  | (8) |
|  | (9) |
|  | (10) |
|  | (11) |
|  | (12) |
|  | (13) |
| |  |  | | --- | --- | |  | (3) | | (14) |
| *Oakl*=*σvl*, *M*(*Oakl*)= *mj* | (15) |
|  | (16) |
|  | (17) |
|  | (18) |

第一个阶段实现了最小化所有产品完成时间的目标，第二个阶段实现了最小化平均生产周期（完成时间和开始时间之间的差异）的目标。约束（3）意味着makespan不得小于任何贮箱的完成时间。约束（4）表明，一个贮箱的完成时间等于其最后一次操作的完成时间。 约束（5）是为了优化目标（2）和满足生产需求而必需的对产品起始时间的调整,用下一段图5进行了具体解释。约束（6）定义产品或罐的开始时间，等于其第一次操作的开始时间。约束（7）强调，操作的完成时间不能小于其起始时间和加工时间之和。 约束（8）意味着，如果两个操作在同一台机器上连续加工，后者的开始时间不能小于前者的完成时间和在机器上的设置时间。约束（9）强制执行，两个连续操作的开始时间之间的间隔时间不能小于前操作的加工时间， *t*是同一产品前一次操作的返工时间。约束（10）和（11）表示两种时间的两种分布，前者表示一个操作的加工时间遵循截断的正态分布，后者表示两台机器之间的传输时间遵循均匀分布。约束（12）定义了加工时间与运输时间之间的相关性，x与X的比值在一个小范围内变化。 约束（13）和（14）表明，在任何时间点，每台机器只能被一个操作占用，一个操作只能被一台机器加工，直到完成。 约束（15）、（16）、（17）和（18）显示了嵌套约束，这意味着只有在满足（16）、（17）和（18）时，（15）才为真。

问题的目标是u个贮箱的最小完成时间和最小的平均生产周期。这两个目标具有相似的特点，可以在一定程度上同时进行优化，同时考虑分选产品对设备利用的影响，这是提高生产系统效率的主要手段。但是，当优化完成时，通过调整第一操作的开始时间，可以进一步优化第二目标。调整的效果可以用图5中的一个简单的情况来说明。在图中，作业的开始时间也是第一次操作的开始时间，不能小于属于另一个作业的前一个操作的完成时间，因为它们共享约束（8）表示的相同机器。有两个作业(J1和J2)，每个作业分别在M1和M2的机器上处理两个操作。 设 *Saj*, *Eaj*, *taj* 分别为*j* of job *a* (*Oaj*)开始时间、结束时间和运行时间. 然后，该作业的生产周期可以表示为 *Ca* = *Ea*2-*Sa*1. Generally, *Sa*1=0, *Sa*2=*Ea*1 (*a*=1). When *a*>1, *Sa*1=*Ea*-1,1+*s*1, *Sa*2= *max* (*Ea*1, *Ea*-1,2+*s*2), when *Ea*-1, 2+*s*2< *Ea*1, *Sa*2= *Ea*1=*Sa*1+*ta*1, *Ea*2=*Sa*2+*ta*2.

然而，当*Ea*-1,2+*s*2> *Ea*1, *Sa*2= *Ea*-1,2+*s*2, *Ea*2=*Sa*2+*ta*2, 如图5b所示, S21=34.75, 而完成的时间是 E22=102.656, 根据定义，J2的生产周期是 E22-S21=67.906 ，这显然不是最优的。 为了缩短J2的周期，建议延迟第一次操作的开始时间，以便其完成时间能够捕捉已经计算出的第二次操作的开始时间，如图5b所示 (E21=S22). 在调整了开始的时间后 J2,, so S21=48.017 and E21=S22, J2的新生产周期可为54.639，这有一个令人鼓舞的结果。通常，如果生产过程完全理想化，这意味着没有设备停机、工件缺陷或其他处理中断，会导致完工延迟， the completion time of the production will maintain unchanged. In fact, however, the production system is fairly complex causing too much uncertainty of both production process and production time. To avoid the risk of delay or larger makespan and production cycle of J2, it is better to set a flexible coefficient as *ρ*(0<*ρ*<1) in constraints (5) for the first operation to make the starting time of J2 between the completion time of the former operation (*O*11) of J1 on the same machine and the latest starting time calculated by the completion time of the operation *O*12 processed on the machine M2 where the next operation *O*22 of J2 is also machined on. As illustrated in figure 5c, the starting time of the first operation of J2 is properly adjusted which gives flexibility for the next operation *O*22.



Figure 5*a*. The original processed Gantt chart



Figure 5*b*. Adjustment of starting time without flexibility after optimization



Figure 5*c*. Adjustment of starting time with flexibility after optimization

4.Improved GA based on *logistic*

4.1 Framework of the algorithm

*Logistic* model is a function to describe the regular growth process of species from a lower number to a dynamic higher number under limited resources , commonly used for the study of biology, demography and economics. The process of species population change can be described by the *logistic* equation:

|  |  |
| --- | --- |
|  | (19) |
|  | (20) |

where *A0* is the initial number of populations; *A*(*t*) is the number of populations at time *t*; *K* is the environmental capacity, which means the upper limit number that a population can reach under natural conditions; *r0* is the initial population growth rate; *r* limited by the population number *A* is the population growth rate over the entire process.

|  |  |
| --- | --- |
|  | (21) |
|  | (22) |

The algorithm flow of improved genetic algorithm based on *logistic* (*LBIGA*) is as the following framework in table 1.

There are five improvement for GA in the algorithm above: the first modification is that the iteration times which works for determining the updating times of the solution population is removed instead the logistic variations is set to control the updating; The second one is an elite population is added to keep the algorithm more active for exploring new solutions. To make the good solution have larger probability to be reserved into the next generation, two authorized solution called the best elite solution *Xtb* and the global best solution called *Xgb* are set. Then an acceptance probability *ptk* is defined to describe the performance for the problem of a solution, which is also the guideline of whether a non-optimal solution is accepted. The last improvement is that the crossing rate and mutation rate are refined as adaptive variables using the formulas (21) and (22).

Table 1. Improved GA based on logistic

|  |  |
| --- | --- |
| 1 | Set the initial size of the species *A*0, initial growth rate *r*0, environmental capacity *K*(*K*>>*A*0), the limit ratio, which is the limit of *A*/*K*, let *A*=*A*0, *r*=*r*0; |
| 2 | Set iteration time *t*=1, initialize the parameters of genetic algorithm: upper limit of variation rate *Pm0*, upper limit of the crossing rate *Pc0*, the selection rate *sr*, the population size and the elite population size, [\*] is the symbol of roundup to get the smallest integer greater than itself; |
| 3 | Initialize *m* solutions (chromosome)for the problem. Then calculate the fitnessof all solutions and select the elite chromosome *Xte* from *Xt* by using the roulette wheel selection approach. Define the best solution of the elite chromosome at *t* iteration *Xtb* = *Xt*, argmax(*f*(*Xte*)) and the global best solution of the population *Xt* until *t* iteration *Xgb* = *Xt*, argmax(*f*(*Xt*)); |
| 4 | While A≤η×K |
| 5 | Set k = 1; |
| 6 | While k < m |
| 7 | Calculate the selection probability of chromosome *Xtk* which can be |
| 8 | If *ptk* > max (*sr*, *A*/*K*) |
| 9 | *Xtb* = *Xtk*, *ftb* = *ftk* |
| 10 | If *ftk­*>*fgb* |
| 11 | *Xgb* = *Xtk*, *fgb* = *ftk* |
| 12 | k=k+1 |
| 13 | Update the population *Xt* with variation and crossing, then select the new elite chromosome *Xe*. |
| 14 | t=t+1 |
| 15 | Update *A* and *r* according to formulas (19) and (20). Update Pc and Pm by formulas (21) and (22). |
| 16 | Output the last solution *Xgb* |

4.2 Coding rules for assembly scheduling problem with nested operations

According to the description in section 3, each tank contains three kinds of parts, and the number of each kind depends on the tank model. For easy calculation of different parts, the tanks are coded in real numbers within this paper.

Take three tanks in two kinds represented with *A*, *A*,and *B* as an example. Suppose that tank *A* requires two parts p1 and p2, and tank *B* requires three parts *p*1, *p*2 and another *p*2, a possible coding of tanks is [0,1,0] meaning a sequence [*ABA*]. Then the coding of parts can be [[0,1],[1,0,1], [1,0]], where the first element with two bits of the coding [0,1] shows the ordering of parts of the first tank *A* as [*p*1*p*2],the middle element with three bits [1,0,1] represents the part sequence of tank *B*, and the last [1,0] express an order of [*p*2*p*1] for *A*.

4. 3 Crossover rules for the special scheduling problem

Since the number of tanks and the number of parts are related to their own types, a general crossover will lead to an unfeasible solution, so a crossover rule for the problem is designed.

Let the optimal solution of the chromosomes be *Sb*, solution to be updated is *Si*. Before the crossover, a cross fragment is firstly randomly generated with the length smaller than the chromosome; then, the fragments of the two chromosomes in the same position are exchanged; finally, the non-swappable fragment is updated. By comparing the gene of the exchange fragments of *Sb* and *Si*, the non-exchange fragments updating is illustrated as figure 6.

Suppose that the length of the two chromosomes is 14, and the rearrangement fragments is shaded in the figure. *Si* becomesafter the exchange. By comparing the number of each gene in the crossed fragments, there are one “0”, two “1” and two “2” in *Si*, and there are two “0”, one “1” and two “2” in *Sb*, which mean that after exchanging fragments, the pro-chromosome has an extra “0” and loses a “1”. To repair the chromosome, a “0” is randomly selected from the non-exchange fragments, and is changed as “1”, as shown in figure 6.



Figure 6. Schematic of the crossover process

4. 4 Variation rules

To explore more solution space, three variation rules are designed according to different gene updating methods. The specific approaches are demonstrated in the following paragraphs and figure 7. When the variation operator is needed, an integer will be stochastically selected from  to execute one manner each time.



Figure 7. Variation approaches of the algorithm

(1) Variation by exchanging two points

For the chromosome *Si* to be updated, two genes in different positions are randomly selected as v1=2 and v2=9 in figure 7, if the genes are the same, two new genes are re-selected; Otherwise, exchange genes of the two chromosomes to form a new chromosome as *S*1*i*.

(2) Variation by inserting one gene into a new position

When two different genes have been selected from one chromosome *Si*, the first gene will be removed from its original place v1 and inserted into the new position which is the former position of v2, the new chromosome is shown as *S*2*i*.

(3) Variation by reversing part of the chromosome

Just like the former two, when two positions are acquired, a fragment from v1 to v2 is also formed, then a chromosome can be gotten with the fragment being reversed as shown in figure 7.

5. Numerical experiments

To find out the better values of parameters in the algorithm, the benchmark *FT*20 (20\*5) was used for experimentation; then the algorithm was compared with the original genetic algorithm (*GA*) on other scheduling problems to verify the effect of the improved GA. The test program is implemented in *Python* and the platform is a personal computer, with *Intel*(*R*) *Core* (*TM*) *i*7, *CPU* @ 3.20*GHz*,16.0 *GB* *RAM*.

5.1 Orthogonal experimental design

The key parameters of the proposed algorithm *LBIGA* are: initial number of species *A*0, environmental capacity *K*, initial growth rate *r*0、the limit approximation parameter *η*、upper limit of cross probability *Pc0*, upper limit of the probability of variation *Pm0*. In the smaller case of *A, sr* can more determine the probability of the algorithm's acceptance of the non-optimal solutions, a smaller *sr* makes the algorithm more inclusive to accept the not so good ones. The four parameters *K*, *A*0, *r*0 and *η* together support the main features of the algorithm, and their values also jointly determine the number of iterations. Among them, *r*0 and *η* are set close to the natural circumstance, and *r0* who needs to be a positive growth rate can be valued from 0 to 0.05, *η* can be 0.9 to 0.99. In order to make the algorithm explore as much as possible while keep , the final number of species which is also called *K* in this paper is set as 100 times greater than the initial species size *A*0. *r0* and *η* are assigned to 0.015 and 0.95, and the *A*0 and *K* is variable. The four parameters are combined working together as a population parameter, expressed as *A*0-*K*-*r*0-*η*. To express *A*0-*K*-*r*0-*η* Conveniently in the chart, *A*0 is selected as the representation of *A*0-*K*-*r*0-*η*, where K=100×*A*0 and *Pc0*, *Pm0* are 0.015, 0.95 respectively. The above parameters were all taken at 4 levels (as shown in Table 2). Orthogonal tests with total number of was carried out, 10 independent trials were performed under each combination of parameters.

Table 2. Orthogonal test factors with different levels

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Level | | | |
| 1 | 2 | 3 | 4 |
| *A*0 | 1 | 10 | 100 | 1000 |
| *Pc0* | 0.7 | 0.75 | 0.8 | 0.85 |
| *Pm0* | 0.05 | 0.1 | 0.15 | 0.2 |
| *sr* | 0.6 | 0.7 | 0.8 | 0.9 |

The subjects were the benchmark *FT*20 problem, and the evaluation criteria were the average *makespan* and variance during the 10 trials. The relevant orthogonal tables and statistical tables are shown in table 3, where *Ij*/4 at the bottom of the table represents the mean of the indicator *Avg\_C* when one of the parameters (*A*0-*K*-*r*0-*η*, *Pc0*, *Pm0* and *sr*) is at the first level while others are at different levels, *IIj*/4 expresses the situation that one parameter is at second level, and so on. *Rj* is the range of the average value of four levels.

Table 3. Orthogonal tables for FT 20 problems and the test results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Parameter  Trial number | Level | | | | | |
| 1 | 2 | 3 | 4 | *Avg*\_*C* | *VAR* |
| 1 | 1 | 1 | 1 | 1 | 1220.1 | 972.49 |
| 2 | 1 | 2 | 2 | 2 | 1240.6 | 1643.04 |
| 3 | 1 | 3 | 3 | 3 | 1237.4 | 1633.84 |
| 4 | 1 | 4 | 4 | 4 | 1238.3 | 1881.41 |
| 5 | 2 | 1 | 2 | 3 | 1252.1 | 1167.49 |
| 6 | 2 | 2 | 1 | 4 | 1246.8 | 1110.56 |
| **7** | **2** | **3** | **4** | **1** | **1205.4** | **903.44** |
| 8 | 2 | 4 | 3 | 2 | 1227.1 | 2089.69 |
| 9 | 3 | 1 | 3 | 4 | 1246.7 | 1850.61 |
| 10 | 3 | 2 | 4 | 3 | 1224.3 | 982.81 |
| 11 | 3 | 3 | 1 | 2 | 1238.4 | 1315.64 |
| 12 | 3 | 4 | 2 | 1 | 1239.7 | 1516.41 |
| 13 | 4 | 1 | 4 | 2 | 1227.7 | 2097.61 |
| 14 | 4 | 2 | 3 | 1 | 1225.4 | 1712.44 |
| 15 | 4 | 3 | 2 | 4 | 1221.3 | 1114.61 |
| 16 | 4 | 4 | 1 | 3 | 1247.2 | 987.16 |
| *Ij*/4 | 1234.1 | 1236.65 | 1238.125 | 1238.275 |  |  |
| *IIj*/4 | 1232.85 | 1234.275 | 1238.425 | 1233.45 |  |  |
| *IIIj*/4 | 1237.275 | 1225.625 | 1234.15 | 1240.25 |  |  |
| *IVj*/4 | 1230.4 | 1238.075 | 1223.925 | 1238.275 |  |  |
| *Rj* | 6.875 | 12.45 | 14.5 | 6.8 |  |  |

According to the results of mean and range above, factor 3 (upper limit *Pm*0of variation) has a greater impact on the experiment. According to the mean of each level, the corresponding utility curve can be made. Hence, the effects of each factor (parameter) on *Avg\_C* is shown in Figure 8, similarly, the effect of each factor (parameter) on VAR is shown in Figure 9. Taken together, the results of test 7 show the best, thus the parameters are selected as, *Pc*0=0.8, *Pm*0=0.2, *sr*=0.6.



Figure 8. Utility of different factors (parameters) on the average production cycle



Figure 9. Utility of different factors (parameters) on the variance of production cycle

5.2 Validation of effectiveness and superiority

To validate the utility of, *Pc*0=0.8, *Pm*0=0.2, *sr*=0.6, different scheduling problems are solved with the proposed algorithm and the original GA. Parameters in *GA* are set as from the document, where *popsize* = 20, *Pc*=0.85, *Pm*=0.05, the maximum number of iterations is 500. The comparison metrics are the average value *Avg*\_*C*, variance *VAR* of optimal makespan, average running time *T* in 10 experiments.

The comparison experiments were firstly carried out on benchmarks of FT06 and FT10, and the results are shown in Table 4.

Table 4. Comparison of the two *benchmarks* experiments

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P | Theoretical optimal solution | Comparison of metrics with *LBIGA* and *GA* | | | | | |
| *Avg*\_*C*  (*GA*) | *Avg*\_*C*  (*LBIGA*) | *VAR*  (*GA*) | *VAR*  (*LBIGA*) | *T*(*GA*) | *T*(*LBIGA*) |
| *FT*06 | 55 | 56.2 | 55.6 | 1.29 | 0.81 | 15.49*s* | 8.86*s* |
| *FT*10 | 930 | 1245 | 1103 | 1878.56 | 1437 | 58.71*s* | 29.32*s* |

Then comparison test of random cases were performed, the problems were of different number of jobs *n* and different number of operations *m*, the operating time *t* was randomly generated from integers among 1 to 100. Each operation is machined at one or two parallel machines. The relevant results are shown in Table 5. To compare the results of the two algorithms intuitively, the optimal results of the two were shown as a ratio of them. For example, “1.00” in the table shows an equal value of two algorithms. A ratio less than 1 means that the *LBIGA* is better than *GA*.

Table 5. Comparison of randomly generated case trials

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *n* | *m* | Ratio of metric values calculated with *LBIGA* and *GA* | | |
| Average optimal solution | Variance | Runtime |
| 6 | 6 | 1.00 | 2.32 | 0.74 |
| 6 | 10 | 0.97 | 0.01 | 0.756 |
| 10 | 6 | 0.997 | 0.00 | 0.74 |
| 10 | 10 | 0.99 | 0.478 | 0.68 |
| 20 | 6 | 0.997 | 0.23 | 0.72 |
| 20 | 10 | 0.96 | 1.38 | 0.72 |

From the comparison results in table 4 and table 5, it can be seen that the average solution time of *LBIGA* is shorter than that of *GA,* that is, the solution efficiency is higher, the logistic design speeded up the iteration speed of genetic algorithms. In the random test cases, the variance ratio only in 6\*6 problem and 20\*10 problem is greater than 1, that means, the robustness of *LBIGA* is overall better than GA.

5.3 ASPs with nested operations

The key parameters in the tank welding workshop scheduling problem are shown in section 3. To solve the model of the problem and verify the performance of the proposed algorithm, 10 cases were generated with random. Still, GA was applied on the cases as a control group like before while the metrics of the tests are the makespan and the mean cycle time of tanks. To make the random cases of the model representative, some parameters were placed approaching to the cases in the real-world scenario while other parameters were endowed stochastically.

The number of tank type *Ɲ* was provided as 3; the number set of each tank *ui* was; number of machine types**was; number of parallel machines was; the number of three kinds of parts in the tank *h* was ; the number of operations for part *Hv* before assembly *L*=**, hence the processing route of each part can be denoted as a random order of**kinds ofmachines; The assembly operation was denoted as *A*, and the processing position was *M*(*A*) whose serial number can be the largest as ; the given machining time of each operation was *X*, which was theoretically an integer ranging from 1 to 30 while the machining time of *OaA* was an integer ranging from 20 to 30; the ratio of the transit time to the machining time**was a random number between 0.01 and 0.05; the rework operation was set as the first operation of the third part, the probability *γ* was a random number between 0.03 and 0.05. The values for parameters above are shown as table 6.

Table 6. Set of values for parameters in assembly scheduling problem with nested operations

|  |  |
| --- | --- |
| Parameter | Value or set of values |
| *Ɲ* | 3 |
| *ui* |  |
|  |  |
|  |  |
| *hiv* |  |
| *X* | is an integer |
|  |  |
| *γ* |  |
| *M*(*A*) |  |
| *o*(*OaA*) | is an integer |

Based on the parameters above, 10 cases were generated. To explain the problem clearly, parameter *Ɲ*, χ, *hiv* and *ψv* are illustrated in one of the generated cases as table 7, table 8, table 9 and table 10. In this case, there are three tanks, the number of which are 6,8,10 respectively. For the first kind of tank , that is *i*=1, the number of three kinds of parts are 2,3,4 respectively.means that there are 5 kinds of machines, which is also the number of operations for parts. From table 9, *M*(*A*) can be 8 for the number of total machines before assembling are 7. These parameters together complicate the problems making it hard to model and state in a natural manner.

Table 7. *Ɲ=*3

|  |  |  |  |
| --- | --- | --- | --- |
| Tank |  |  |  |
| Count | 6 | 8 | 10 |

Table 8. The number of each kind of part and tank

|  |  |  |  |
| --- | --- | --- | --- |
| Part  Tank |  |  |  |
| (*i*=1) | 2 | 3 | 4 |
| (*i*=2) | 2 | 2 | 4 |
| (*i*=3) | 3 | 3 | 3 |

Table 9. 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Machine |  |  |  |  |  |
| Count | 2 | 2 | 1 | 1 | 1 |

Table 10. Processing route of the parts

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Part  Tank | *ψv* | | | | |
| *σv*1 | *σv*2 | *σv*3 | *σv*4 | *σv*5 |
| (*i*=1) |  |  |  |  |  |
| (*i*=2) |  |  |  |  |  |
| (*i*=3) |  |  |  |  |  |

To observe the effects of the two algorithms, one original solution of the problem solution was generated without optimizing, then LBIGA and GA were applied to optimize the problem based on the original solution. The results are shown in table 10, where *Original\_C* and *Original\_CT* represent the makespan and the mean cycle time of the production according to the original solution respectively, *Op\_rate* is the index of how well the original values (*Original\_C* and *Original\_CT*, expressed as *L\_original*) were optimized by algorithm *L\_a*. The computing technique is defined as formula (22).

|  |  |
| --- | --- |
|  | (22) |

Table 10. Improvement of mean values of the metrics on randomized cases with *LBIGA* vs. *GA*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Case | *Original\_C* | *Original\_CT* | *Op\_rate* | | | |
| *LBIGA*\_*C* | *LBIGA*\_*CT* | *GA*\_*C* | *GA*\_*CT* |
| 1 | 2799.48 | 312.6 | 11.43% | 12.17% | 8.44% | 8.24% |
| 2 | 2833.64 | 333.2 | 10.90% | 17.11% | 8.38% | 17.07% |
| 3 | 2744.32 | 306.58 | 6.58% | *9.16%* | 5.71% | *9.94%* |
| 4 | 2812.54 | 315.24 | 10.42% | 12.85% | 7.93% | 11.67% |
| 5 | 2836.35 | 316.25 | 7.26% | *10.92%* | 5.95% | *10.97%* |
| 6 | 2825.83 | 307.43 | 6.03% | 8.84% | 4.83% | 7.18% |
| 7 | 2814.52 | 317.83 | 8.03% | 13.69% | 7.57% | 10.54% |
| 8 | 2826.73 | 313.27 | 8.16% | 11.51% | 8.04% | 11.37% |
| 9 | 2791.74 | 304.96 | *7.29%* | 10.62% | *7.31%* | 10.45% |
| 10 | 2801.63 | 312.27 | 8.06% | 11.55% | 6.80% | 8.61% |

Comparative results in table 7 show that *LBIGA* has better performance than *GA* in almost all cases: only in case 3 and case 5, the mean production cycle with *GA* was improved more than that with *LBIGA*. As for completion time, except for case 9, *LBIGA* performed better than *GA*.

6. Conclusion

In this paper assembly scheduling problems (*ASPs*) with complex nested operations are introduced, and an improved genetic algorithm based on *logistic* model is proposed for minimizing the completion time and the mean production cycle of the products. The ASP model is based on the welding process of rocket tanks, considering the special constraints like multi-level nesting, random rework or dynamic machining time existing in the actual processing process. There are numerous parameters and variables in the model which exactly construct the key characteristics of the rocket tanks production. Besides, to efficiently address the problem, the *logistic* model was introduced and utilized to improve the genetic algorithm (*GA*) as logistic-based-improved-genetic-algorithm (*LBIGA*). By different numerical experiments, *LBIGA* is validated to has higher effectiveness and superiority, showing the success of the logistic application. Also, there supposed to be more details to be considered in the ASPs of the rocket tank production which are difficult to be elaborated with math formulas and may be provided by other methods like simulation technology. Nevertheless, this paper is a new try to build mathematical models aiming to depict the picture of the complex production and deliver an idea for the researchers of both academia and industry.

In addition to rocket tanks, products of other complex assemblies like airplane, ship, high-speed rail etc., also have the characteristics of single or small-batch production and involving many parts. To dispatch the parts and the machines needed, mathematical modeling and appropriate approaches are necessary, the key is to acquire the specific conditions and objectives that should be translated into mathematical language. Based on the current approaches and models, new effective methods can be developed as to tackle the problems of real scenarios.