混合离散蛙跳算法求解柔性装配系统调度问题

李晓玲[1,2†], 冯彦翔[2], 张广辉[3], 段浩浩[1]

(1. 长安大学 电子与控制工程学院, 陕西 西安 710064; 2. 西安交通大学 机械制造系统工程国家重点实验室, 陕西 西安 710049;
3. 河北农业大学 信息科学与技术学院, 河北 保定 071001)

摘要: 本文主要研究不含中间缓冲区的柔性装配系统(FAS)的优化调度问题, 其中当工件竞争使用有限的生产资源时, 不合理的资源分配会导致系统死锁(deadlock). 针对易死锁(deadlock-prone)FAS的优化调度问题, 本文采用Petri网建模, 提出了一种混合离散蛙跳算法(HDSFLA)以最小化最大完工时间(makespan). 首先, 提出了一种新的编码解码方法, 其中一个个体编码为一个包含全部工件加工信息的变迁序列, 可解码为一个工件–工序序列; 其次, 为了保证种群中个体的可行性, 提出了一个个体修正算法和基于最早引发时间的改进个体修正算法, 从而将不可行个体修复为可行个体; 然后, 结合编码特征设计了用于生成新个体的交叉操作; 最后, 为了平衡算法的全局搜索和局部开发能力, 设计了一个基于交换和插入算子的局部搜索策略. 通过不同规模算例上的仿真实验和算法对比分析, 验证了HDSFLA的有效性.

关键词: 柔性装配系统; 死锁; Petri网; 调度; 混合离散蛙跳算法

引用格式: 李晓玲, 冯彦翔, 张广辉, 等. 混合离散蛙跳算法求解柔性装配系统调度问题. 控制理论与应用, 2025, 42(4): 816 – 826

DOI: 10.7641/CTA.2023.30331

Hybrid discrete shuffled frog leaping algorithm for the scheduling problem of flexible assembly systems

LI Xiao-ling[1,2†], FENG Yan-xiang[2], ZHANG Guang-hui[3], DUAN Hao-hao[1]

(1. School of Electronics and Control Engineering, Chang'an University, Xi'an Shaanxi 710064, China;
2. The State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University, Xi'an Shaanxi 710049, China;
3. College of Science and Technology, Hebei Agricultural University, Baoding Hebei 071001, China)

Abstract: This paper addresses the scheduling problem in flexible assembly systems (FAS) without intermediate buffers. In such systems, deadlock may occur when multiple jobs compete for limited shared resources under inappropriate allocation. To solve the scheduling problem of deadlock-prone FAS, Petri nets are used to model it, and a hybrid discrete shuffled frog leaping algorithm (HDSFLA) is proposed to minimize the maximum completion time, i.e., makespan. Firstly, a novel encoding and decoding method is proposed, in which an individual is encoded as a complete transition sequence and can be decoded into a sequence of jobs and operations. Then, in order to guarantee the feasibility of each individual, an individual modification method and an improved individual modification method based on the-earliest-firing-time are developed, through which an unfeasible solution can be repaired into a feasible one. Furthermore, based on the characteristics of the encoding method, a crossover operation is designed to generate new individuals for the next generation. In addition, to balance the global exploration and local exploitation capabilities of the proposed algorithm, a local search method based on swap and insert operators is developed and imbedded into the algorithm. Experimental tests on different instances and comparisons with other algorithms are conducted to demonstrate the effectiveness of HDSFLA.

Key words: flexible assembly systems; deadlock; Petri nets; scheduling; hybrid discrete shuffled frog leaping algorithm

Citation: LI Xiaoling, FENG Yanxiang, ZHANG Guanghui, et al. Hybrid discrete shuffled frog leaping algorithm for the scheduling problem of flexible assembly systems. *Control Theory & Applications*, 2025, 42(4): 816 – 826

收稿日期: 2023−05−17; 录用日期: 2023−12−19.
†通信作者. E-mail: xiaolingli@chd.edu.cn; Tel.: +86 15529569712.
本文责任编委: 丛爽.
国家自然科学基金项目(62103062, 62003258), 长安大学中央高校基本科研业务费专项资金项目(300102322101)资助.
Supported by the National Natural Science Foundation of China (62103062, 62003258) and the Fundamental Research Funds for the Central Universities, CHD (300102322101).

1 引言

生产调度对于提升制造企业的运行效率起着至关重要的作用. 装配制造系统的调度问题, 也称装配调度问题(assembly scheduling problem, ASP), 由于其在实际生产中的广泛应用成为近年来的研究热点[1–3]. 不同类型的ASP, 如装配流水车间调度问题(assembly flowshop scheduling problem, AFSP)[4–6]、分布式装配流水车间调度问题(distributed AFSP, DAFSP)[7–10]、装配作业车间调度问题(assembly jobshop scheduling problem, AJSP)[11–12]、柔性装配作业车间调度问题(flexible AJSP, FAJSP)[13–15] 等, 吸引了研究者们的广泛关注和研究. 相较于一般的车间调度问题, ASP不仅要考虑子组件或半成品的加工, 还要考虑产品的组装, 其求解更为复杂.

现有关于ASP的研究通常假设两个相邻的加工机器之间存在着容量无限大的中间缓冲区, 且研究成果主要集中于AFSP[4–6]和DAFSP[7–10], 关于AJSP[11–12]和FAJSP[13–15]的研究还较少. Shi等[11]提出了一种混合遗传算法来解决以最小化makespan为目标的AJSP. Cheng等[12]针对一个考虑了材料完整性和装配顺序约束的AJSP, 建立了一种增强模拟退火算法(enhanced simulated annealing, ESA), 以最小化总完工时间和总库存时间. Zhang和Wang[13]针对具有序列相关准备时间、工件共享和动态事件的FAJSP, 提出了分别基于最早完工时间、工件最多剩余加工时间和最小工作量机器的调度规则, 以解决调度和重调度问题来最小化makespan. Wu等[14]研究了一个以总提前/延迟时间和总花费为优化目标的分布式FAJSP, 提出了一种改进差分进化模拟退火算法(improved differential evolution simulated annealing algorithm, IDESAA), 其中设计了交叉、变异和选择算子用于种群更新, 并设计了一种模拟退火算法用于搜索最佳Pareto解. Zhang等[15]研究了一个多目标FAJSP, 其中考虑了序列相关准备时间和工件在不同机器之间的传输时间, 提出了一种多目标分布式蚁群方法(distributed ant colony system, DACS)来优化总工作量、总延迟和makespan. Xu 和Nagi[16]研究了具有树结构和并行机的AJSP, 并开发了一种拉格朗日松弛(lagrangian relaxation, LR)方法以最小化makespan.

上述文献虽然考虑了多种生产约束, 但都没有考虑阻塞约束, 即所研究的AJSP和FAJSP均假设相邻的两个加工机器之间存在着容量无限大的中间缓冲区. 而在许多实际生产系统中, 如电子制造车间、自动化生产单元、钢铁工业、化工和制药厂等, 由于生产环境的空间约束或产品生产技术要求, 中间缓冲区是容量有限的或者不存在的, 因此含有阻塞约束(零中间缓冲区)的制造调度问题近年来也吸引了越来越多研究者的关注[7–9, 17–22].

阻塞装配作业车间调度问题(blocking AJSP, BAJSP)可以在实际工业生产中找到许多原型, 如电子制造工厂、汽车制造厂等, 然而, 目前围绕BAJSP的研究成果是极少的[17]. 这是因为在BAJSP中, 由于生产资源 (如加工机器、运输设备等) 是有限的, 如果资源得不到合理分配, 则可能出现死锁现象, 即系统是易死锁的(deadlock-prone), 死锁将会导致整个或部分系统处于无限期的阻塞, 无法完成加工任务[23–24]. 死锁通常会导致系统长时间停工、一些关键或昂贵资源的低使用率以及生产效率低等结果, 使得系统的关键性能指标受到严重影响. 在求解易死锁BAJSP时, 需要同时考虑死锁避免和调度目标的优化, 这使得调度算法的设计变得更加复杂. 现阶段只有文献[17]研究了一类BAJSP的变体, 提出了一种混合粒子群优化算法(hybrid particle swarm opitimization, HPSO), 以最小化makespan.

文献[17]虽然针对一类易死锁BAJSP进行了研究, 但是并没有考虑工件具备柔性加工路径的情况. 因此, 为此类问题设计可行且高效的调度算法仍有待研究. 本文针对一个FAJSP的变体, 即一类含有并行机和批量工件、不含中间缓冲区、工件具备柔性加工路径的易死锁制造装配系统, 简称易死锁柔性装配系统(flexible assembly systems, FAS), 进行建模和调度问题求解的研究. 在建模方面, 考虑到通过数学公式很难描述死锁且建立不等式约束来避免死锁是很困难的, 以及Petri网在为制造系统建模、表征死锁、分析系统运行行为等方面表现出的优异性能[23–24], 本文采用Petri网对FAS建模; 在问题求解方面, 考虑到蛙跳算法(shuffled frog leaping algorithm, SFLA) 具有参数少、结构简单、易实现等特点, 同时具备局部搜索和全局信息交换等特点[25], 本文采用了SFLA并结合问题特征设计了混合离散蛙跳算法(hybrid discrete SFLA, HDSFLA)来使其能够有效地求解易死锁柔性装配系统的调度问题. 仿真实验和算法比较验证了HDSFLA的有效性.

2 问题描述

2.1 柔性装配系统

本文研究的FAS可描述为: FAS 包含 $\mathcal{R}$ 类机器, 记作$R=\{r_m|m\in Z_{\mathcal{R}}\}$; 能够完成$\mathcal{N}$类工件的加工, 记作$J=\{J_n|n\in Z_{\mathcal{N}}\}$, 其中为方便起见, $Z_{\mathcal{R}}$和$Z_{\mathcal{N}}$分别代表$Z_{\mathcal{R}}=\{1,2,\cdots,\mathcal{R}\}$和$Z_{\mathcal{N}}=\{1,2,\cdots,\mathcal{N}\}$, 下同. 令$C(r_m)$表示$r_m$的容量, 即具有相同加工能力的机器$r_m$的个数. 令$\varphi_n$表示待加工的$J_n$类工件的个数, 系统中待加工工件的总数为$\Phi=\sum\limits_{n\in Z_{\mathcal{R}}}\varphi_n$. 此外, J_1类工件的工件序号表示为$1,2,\cdots,\varphi_1$; J_2 类工件表示为$\varphi_1+1,\varphi_1+2,\cdots,\varphi_1+\varphi_2$; J_3类工件表示为φ_1+

$\varphi_2+1, \varphi_1+\varphi_2+2, \cdots, \varphi_1+\varphi_2+\varphi_3$; 等.

在FAS中, 一个产品是由多个工件经过一系列的加工和组装操作之后完成生产的. 在一个加工操作中, 工件只是利用机器进行加工; 而在组装操作中, 两个或多个工件组装成一个新的半成品或最终产品. J_n类工件的一条加工路径可以表示为一个操作序列, 工件可能有多条加工路径, 即加工路径具有柔性. 令Ω_n表示J_n类工件的加工路径集合, 其中: $\omega_i = o_{i,1}o_{i,2}\cdots o_{i,l}\cdots o_{i,L_i}(\omega_i \in \Omega_n)$为$J_n$类工件的一条加工路径, $o_{i,l}$为ω_i上的第l个操作, L_i为ω_i的长度. 本文中只考虑了同类工件加工路径长度相同的情况. 令$R(o_{i,l})$表示完成操作$o_{i,l}$所需要的资源, 于是一条加工路径与一个资源序列对应, ω_i可以由资源序列$R(o_{i,1})R(o_{i,2})\cdots R(o_{i,L_i})$决定. 对于同类工件不同加工路径上的两个操作, 例如, $o_{i,l}$和$o_{i',l}$, 如果$R(o_{i,l})=R(o_{i',l})$, 那么$o_{i,l}$和$o_{i',l}$表示同样的操作, 即ω_i和$\omega_{i'}$上的第l个操作是相同的. 在本文中, 假设FAS中所有操作所需的加工时间是预先确定且已知的, 令$d_{i,l}$表示完成$o_{i,l}$所需的加工时间.

FAS中其他约束条件为: 1) 不存在中间缓冲区; 2) 所有机器在0时刻可用, 所有工件在0时刻可开始加工; 3) 每个机器同一时刻至多只能加工1个工件; 4) 每个工件任一时间最多只能在1个机器上加工; 5) 任一操作一旦开始不允许中断; 6) 机器的准备时间忽略不计(或包含在加工时间内).

2.2 FAS的Petri网调度模型

2.2.1 Petri网基本定义

Petri网是一个三元组$N=(P,T,F)$, 其中: P是库所集合, T是变迁集合, $F\subseteq(P\times T)\cup(T\times P)$为有向弧集, 分别由圆圈、方框和带箭头的连接线表示; 且有$P\neq\varnothing, T\neq\varnothing, P\cap T=\varnothing$. 对于$x\in P\cup T$, ${}^\bullet x=\{y\in P\cup T|(y,x)\in F\}$表示$x$所有输入元素的集合, $x^\bullet=\{y\in P\cup T|(x,y)\in F\}$表示$x$所有输出元素的集合.

N的状态是一个映射$M: P\to Z=\{0,1,2,3,\cdots\}$. 给定库所$p\in P$和状态$M$, $M(p)$表示在M下p中托肯(token)的个数. N的任意一个状态M可以表示为$M=\sum\limits_{p\in P}M(p)\cdot p$. 一个具有初始状态$M_0$的Petri网称为标记Petri网, 记作$(N,M_0)$.

变迁t在状态M下是使能的(enabled), 当且仅当$\forall p\in{}^\bullet t, M(p)>0$, 记作$M[t>$. 在状态$M$下的使能变迁$t$可以引发, 使得系统到达新状态$M'$, 记作$M[t>M'$, 其中$M'(p)=M(p)-1, \forall p\in{}^\bullet t\backslash t^\bullet$; $M'(p)=M(p)+1, \forall p\in t^\bullet\backslash{}^\bullet t$; 否则$M'(p)=M(p)$. 一个变迁序列$\beta=t_1t_2\sum t_K$如果满足$M_k[t_k>M_{k+1}, k\in Z_K$, 且$M_1=M$, 则称其在$M$下是可行(feasible)的.

2.2.2 FAS的Petri网模型

为了建立FAS的Petri网模型, 需要先后为FAS中的加工路径、资源的请求和释放建模. 读者可参考文献[26]来更好的了解Petri网建模.

首先, J_n类工件的一条加工路径$\omega_i=o_{i,1}o_{i,2}\cdots o_{i,l}\cdots o_{i,L_i}$可以建模为Petri网中的一条库所变迁路径(place transition path, PT–路径), 记作$\mathrm{PT}_i=p_{n,0}t_{i,1}p_{i,1}t_{i,2}\cdots t_{i,l}p_{i,l}t_{i,l+1}\cdots p_{i,L_i}t_{i,L_i+1}p_{n,L_i+1}$, 其中$p_{i,l}(l\in Z_{L_i})$是一个操作库所, $o_{i,l}$的加工在$p_{i,l}$上进行, 且$p_{i,l}$中托肯的个数表示正在加工的工件个数, 变迁$t_{i,l}$的引发表示$p_{i,l-1}$中操作的结束和$p_{i,l}$中操作的开始. 方便起见, 对每类工件J_n引入两个虚拟的操作库所, 分别用于存储等待加工的工件和已完成加工的工件, 记作$p_{n,0}$和p_{n,L_i+1}, 称为开始库所和结束库所. 工件在存储位置、加工位置之间的移动用库所到变迁以及变迁到库所的有向弧表示. 对于一个操作库所$p_{i,l}$, 如果$|p_{i,l}^\bullet|>1$, 即$p_{i,l}$有两个及以上的输出元素, 则称$p_{i,l}$为分支库所, 其中$|p_{i,l}^\bullet|$表示$p_{i,l}^\bullet$中元素的个数. 给定一个变迁$t_{i,l}$, 令${}^{(p)}t_{i,l}$表示${}^\bullet t_{i,l}$中的操作库所构成的集合. 如果$|{}^{(p)}t_{i,l}|>1$, 即${}^{(p)}t_{i,l}$中包含的元素个数大于1, 则称$t_{i,l}$为组装变迁. 一个组装变迁的引发表示系统中相应组装操作的开始. 由于FAS中不同的加工路径可能共享一些相同的操作, 相应的, 其对应的PT–路径会共享对应的库所、变迁和有向弧.

在FAS的Petri网模型中, r_m类机器用一个资源库所r_m表示. 资源库所r_m中托肯的个数表示可用机器的数量, 在系统的初始状态M_0下, r_m中托肯的个数$M_0(r_m)$即为系统中r_m类机器的总个数. 当操作库所$p_{i,l}$上的加工需要使用资源r_m, 则添加从r_m到$t_{i,l}={}^\bullet p_{i,l}$和从$t_{i,l+1}=p_{i,l}^\bullet$到$r_m$的有向弧表示资源的需求和释放.

整个FAS的Petri网模型为$(N,M_0)=(P,T,F,M_0)=(P_\mathrm{O}\cup P_\mathrm{S}\cup P_\mathrm{E}\cup P_\mathrm{R},T,F,M_0)$, 其中$P_\mathrm{O}$, P_S, P_E和P_R表示操作库所、开始库所、结束库所和资源库所的集合. 初始状态M_0定义为$M_0(p_{n,0})=\varphi_n$, $\forall p_{n,0}\in P_\mathrm{S}$; $M_0(p)=0, \forall p\in P_\mathrm{O}\cup P_\mathrm{E}$; $M_0(r_m)=C(r_m), \forall r_m\in P_\mathrm{R}$.

对于操作库所$p_{i,l}$中的任意一个托肯, 其在$p_{i,l}$中的停留时间至少为$d_{i,l}$, 即完成$o_{i,l}$所必需的加工时间. 当FAS完成加工后, 其Petri网模型对应地到达终止状态, 记作M_f, 且有$M_\mathrm{f}(p)=0, \forall p\in P_\mathrm{O}\cup P_\mathrm{S}$; $M_\mathrm{f}(p_{n,L_i+1})=M_0(p_{n,0}), \forall p_{n,L_i+1}\in P_\mathrm{E}$; $M_\mathrm{f}(r_m)=C(r_m), \forall r_m\in P_\mathrm{R}$.

给定一个FAS及其Petri网模型, 一个满足$M_0[\alpha>M_\mathrm{f}$的变迁序列α称为该FAS的一个调度序列. 本文所研究的FAS调度问题为寻找一个调度序列并使得其对应的完工时间尽可能地小.

2.2.3 基于Petri网模型计算完工时间

给定一个FAS及其Petri网模型(N, M_0), 令$\pi=\pi_1, \pi_2, \cdots, \pi_{H-1}, \pi_H$表示一个满足$M_0[\pi > M_{\rm f}$的变迁序列, 其中$H$为$\pi$中变迁的个数. 对于该调度序列中任意一个变迁$\pi_h(h \in Z_H)$, 假设$\pi_h = t_{i,l}$且其为$\pi$中第$a$个$t_{i,l}$, 为表示更清楚, π_h也可记为$\pi_h[t^a_{i,l}]$, π_h的最早引发时间记作$f(\pi_h) = f(\pi_h[t^a_{i,l}])$. 根据$\pi$中变迁的顺序, π_h 的引发时间不能早于π_{h-1}的引发时间, 因此有$f(\pi_h) \geqslant f(\pi_{h-1})$. 假设$\pi_{h-1}$对应于$\pi$中的第$a'$个$t_{i',l'}$, 那么$f(\pi_h) \geqslant f(\pi_{h-1})$可进一步表示为

$$f(\pi_h[t^a_{i,l}]) \geqslant f(\pi_{h-1}[t^{a'}_{i',l'}]),$$

此外, 鉴于$t_{i,l}$的引发对应于操作$o_{i,l}$的开始, 当$t_{i,l}$不是组装变迁时, 即$|^{(p)}t_{i,l}|=1$, $\pi_h=t_{i,l}$还必须在$o_{i,l}$的前一个操作 $o_{i,l-1}$完成之后才能引发, 即 π_h的引发时间还需满足 $f(\pi_h[t^a_{i,l}])\geqslant f(\pi_{h'}[t^a_{i,l-1}]) + d_{i,l-1}$. 因此对于一个非组装变迁, $f(\pi_h[t^a_{i,l}])=\max\{f(\pi_{h-1}[t^{a'}_{i',l'}]), f(\pi_{h'}[t^a_{i,l-1}]) + d_{i,l-1}\}$, $|^{(p)}t_{i,l}| = 1$. 其中由于 $t_{i,0}$ 和 $o_{i,0}$在FAS的Petri网调度模型中不存在, 故有

$$f(\pi_{h'}[t^a_{i,0}]) = 0,\ d_{i,0} = 0.$$

当$\pi_h=t_{i,l}$为组装变迁时, 即$|^{(p)}t_{i,l}|>1$, 它只有在$^{(p)}t_{i,l}$中所有操作库所中的工件准备好组装时才能引发, 即 π_h的引发时间需满足

$$f(\pi_h[t^a_{i,l}]) \geqslant \max\{f(\pi_q[t^a_{x',y'}])+d_{x,y}|p_{x,y}\in{}^{(p)}t_{i,l},\ t_{x',y'}\in{}^{\bullet}p_{x,y}\},$$

其中$t^a_{x',y'}$为π中满足$t_{x',y'}\in{}^{\bullet}p_{x,y}$的第$a$个$t_{x',y'}$. 同时, π_h的引发时间不能早于π_{h-1}的引发时间, 故有

$$f(\pi_h[t^a_{i,l}]) = \max\{f(\pi_{h-1}[t^{a'}_{i',l'}]), \max\{f(\pi_q[t^a_{x',y'}]) + d_{x,y}|p_{x,y}\in{}^{(p)}\ t_{i,l}, t_{x',y'}\in{}^{\bullet}p_{x,y}\}\}.$$

综上, π_h最早引发时间的计算公式为

$$f(\pi_h[t^a_{i,l}]) = \begin{cases} \max\{f(\pi_{h-1}[t^{a'}_{i',l'}]),\ f(\pi_{h'}[t^a_{i,l-1}]) + d_{i,l-1}\}, \\ \qquad\qquad |^{(p)}t_{i,l}|=1, \\ \max\{f(\pi_{h-1}[t^{a'}_{i',l'}]),\ \max\{f(\pi_q[t^a_{x',y'}])+ \\ d_{x,y}|p_{x,y}\in{}^{(p)}t_{i,l}, t_{x',\,y'}\in{}^{\bullet}p_{x,y}\}\},\ |^{(p)}t_{i,l}|>1. \end{cases} \tag{1}$$

π中最后一个变迁π_H的引发时间即为该调度序列对应的完工时间, 记为$C_{\max}(\pi) = f(\pi_H)$.

例 1　考虑一个机器集为$R = \{r_m|m \in Z_5\}$, 工件集为$J = \{J_1, J_2\}$的FAS, 其简单示意图见图 1, 其中每类资源的容量为$C(r_m) = 1, m \in Z_5$, 待加工的工件数量为$\varphi_1 = \varphi_2 = 5$.

在该FAS中, J_1类工件有一条加工路径, $\omega_1 = o_{1,1}o_{1,2}o_{1,3}$, 各操作使用的机器分别为$r_1$, r_2和r_5; J_2类工件有两条加工路径, $\omega_2 = o_{2,1}o_{2,2}o_{2,3}$和$\omega_3 = o_{3,1}o_{3,2}o_{3,3}$, $\omega_2(\omega_3)$ 上各操作使用的机器分别为 r_1, r_4 和 r_5 (r_3, r_4和r_5). 由于$o_{2,2}$和$o_{3,2}$使用的机器满足$R(o_{2,2})=R(o_{3,2}) = r_4$, 因此$o_{2,2} = o_{3,2}$, 在Petri网模型中用同一个操作库所$p_{2,2}$表示. 另外, 由于系统中存在将$J_1$类工件和$J_2$类工件组装成最终产品的组装操作, 因此, $o_{1,3}$, $o_{2,3}$和$o_{3,3}$表示同一个操作, 在Petri网模型中用$p_{1,3}$表示. 系统的Petri网模型如图1所示.

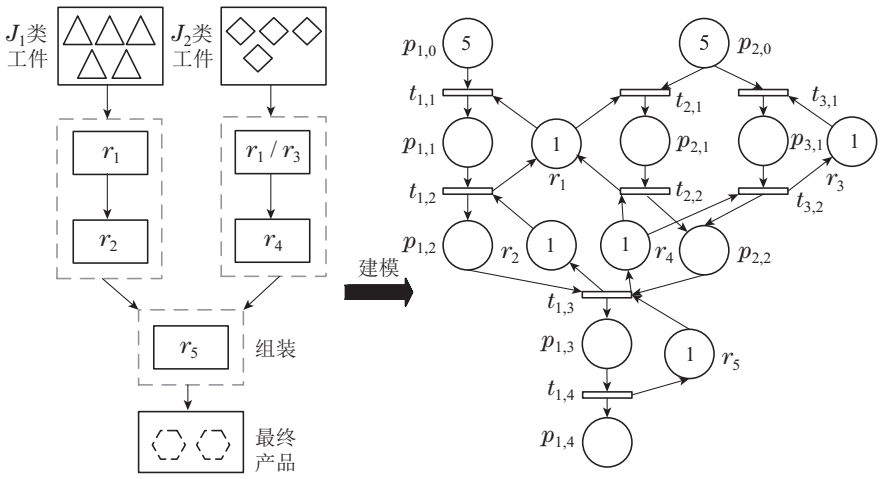


图 1 FAS示意图及其Petri网模型

Fig. 1 Example of an FAS and its Petri net model

3 混合离散蛙跳算法

为了解决FAS的调度问题, 本文提出了HDSFLA以最小化完工时间, 主要包含5个模块: 个体生成与修正、个体解码、模因组划分、模因组内搜索和局部搜索.

3.1 个体生成与修正

3.1.1 编码

给定一个FAS及其Petri网模型, 算法中的一个个体S编码为一个具有重复变迁的完整变迁序列$S=s_1, s_2, \cdots, s_H$, 即S包含从M_0到达$M_{\rm f}$所需的变迁. H的

值取决于系统中待加工的每类工件的个数以及其加工路径(PT–路径)的长度.

由于系统中存在组装操作, 不同类型工件的PT–路径可能拥有相同的子PT–路径, 为了避免该子PT–路径上的变迁出现的次数多于一个可行S所需要的次数, 为PT_i定义了一个用于编码的有效(valid) PT–路径(VPT–路径), 记为VPT_i. 即, 通过删减一些PT–路径上的相同子PT–路径以保证Petri网模型中的变迁在$\mathrm{V–PT}_i$中都只出现一次. 令T_i表示一个集合, 由VPT_i上的变迁构成. 令Θ_n表示由J_n类工件的VPT–路径上的变迁构成的集合, $\Theta_n = \underset{\omega_i \in \Omega_n}{\cup} T_i$.

给定一个含有$\mathcal{N}$类工件的FAS及其Petri网模型, J_n类工件的VPT–路径的长度记为λ_n, 那么一个调度序列的长度为$H = \sum\limits_{n=1}^{\mathcal{N}} \varphi_n \times \lambda_n$. 令算法种群的大小为POP, 本文采用随机方式生成初始种群中的个体, 记作算法1, 详细步骤如下:

步骤 1 令$\Lambda = \varnothing$, $\Lambda_n = \varnothing (n \in Z_{\mathcal{N}})$. 对于$J_n$类工件, 如果该类工件只有一条VPT–路径, 设为VPT_i, 则令$\Lambda_n = \Lambda_n \cup T_i$; 否则, 在该类工件的VPT–路径中任意选择一条, 设为$\mathrm{VPT}_{i'}$, 并令$\Lambda_n = \Lambda_n \cup T_{i'}$; 重复以上操作$\varphi_n$次. 令$\Lambda = \underset{n \in Z_{\mathcal{N}}}{\cup} \Lambda_n$.

步骤 2 令$h = 1$, $S = s_1, s_2, \cdots, s_H$.

步骤 3 随机选择Λ中的一个元素, 如$t_{i,l}$, 令$s_h = t_{i,l}$, $\Lambda = \Lambda \backslash \{t_{i,l}\}$, $h = h + 1$.

步骤 4 若$\Lambda = \varnothing$, 则输出S; 否则返回步骤3.

为方便读者理解, 提供一个编码示例见附录.

3.1.2 个体修正

由于算法1得到的S是随机生成的, 没有考虑系统中的各种生产约束, 如工序的先后顺序和资源容量约束, 以及死锁状态, 因此S可能是不可行的. 为了得到一个可行的调度序列, 需对S进行修正使得S满足$M_0[S > M_f$, 为此, 本文提出一种个体修正(individual modification, IM)方法, 记作算法2(见表1).

给定一个个体S, 修正开始于初始标识$M_1 = M_0$和S中的第1个变迁. 在算法2的第h个修正步骤中, 首先检测s_h是否可行(s_h称为M_h下的可行变迁, 当且仅当s_h在M_h下使能且s_h是安全变迁; s_h引发后得到的新标识M_{h+1}是安全的, 则称s_h为安全变迁), 如果可行则直接引发, 更新标识, 并开始下一步修正; 否则将$s_{h+1}, s_{h+2}, \cdots, s_H$中的第1个可行变迁$s_{h+d}$ $(d > 0)$移到s_h之前, 在M_h下引发s_{h+d}并开始下一步修正. 由于本文考虑的FAS中存在路径柔性, 因此可能出现这样一种情况: 待检测的变迁s_{h+d}是不可行的(即s_{h+d}不使能或者s_{h+d}不安全), 但是$s_{h+d} \in p^{\bullet}_{i,j}$, $|p^{\bullet}_{i,j}| > 1$, 且存在$t \in p^{\bullet}_{i,j}$是可行的, 则称以上这种现象为路径冲突, 即对于正在加工的、具有路径柔性的工件而言, 其当前选择的加工路径是不可行的, 但是存在其他的可行路径, 这时通过更改工件的加工路径可完成第h步的修正. 为了快速地判定一个变迁t在一个给定标识M下是否安全, 算法2中采用了一种具有多项式复杂度的死锁避免策略(deadlock avoidance policy, DAP)[27]来进行判定.

表 1 算法 2 IM算法

Table 1 Algorithm 2 IM algorithm

输入: FAS的Petri网模型(N, M_0), $S = s_1, s_2, \cdots, s_H$
输出: 修正后的可行调度序列S

```
1 令M_1 = M_0;
2 for(h = 1; h ⩽ H; h++)
3   令d = 0;
4   while(s_{h+d}在M_h下不可行)
5     if (s_{h+d} ∈ p•_{i,j} ∧ |p•_{i,j}| > 1)/*s_{h+d}是分支库所p_{i,j}的一个输出变迁*/
6       if (∃t满足t ∈ p•_{i,j} ∧ t ≠ s_{h+d} ∧ t在M_h下是可行的)
7         设s_{h+d} = t_{i,j+1}, t = t_{i',j+1}; /*存在路径冲突, 修改工件的加工路径*/
8         for(y = j + 1; y ⩽ |T_{i'}|; y++)
9           将s_h, s_{h+1}, ···, s_H中的t_{i,y}替换为t_{i',y};
10        end for
11      else
12        d := d + 1;
13      end if
14    else
15      d := d + 1;
16    end if
17  end while
18  if (d = 0)
19    M_h[s_{h+d} > M_{h+1};/*s_h是M_h下的可行变迁*/
20  else
21    M_h[s_{h+d} > M_{h+1}; S := s_1, s_2, ···, s_{h-1}, s_{h+d}, s_h, s_{h+1}, ···, s_{h+d-1}, s_{h+d+1}, ···, s_H;
22  end if
23 end for
24 End
```

在IM算法的第h个修正步骤中, 算法只是在s_h, $s_{h+1}, s_{h+2}, \cdots, s_H$中寻找第1个可行变迁$s_{h+d}$, 或通过改变工件的加工路径使得$s_{h+d}$为可行变迁, 并没有利用与变迁引发时间相关的信息, 因此得到的个体性能可能不佳. 为了保证修正后得到的个体具有良好的性能, 本文还提出了一种改进后的个体修正(improved IM, IIM)算法. 在IIM的第h个修正步骤中, 建立了一种最早引发时间(earliest firing time, EFT)策略来选择M_h下具有最早引发时间的可行变迁. 具体过程为: 首先, 计算M_h下具有最早引发时间的变迁t(当有多个变

迁时, 随机选择一个); 然后, 在$s_h, s_{h+1}, s_{h+2}, \cdots, s_H$中寻找$t$, 并将$s_y = t$移动到$s_h$之前. 如果$s_h$, s_{h+1}, $s_{h+2}, \cdots, s_H$中不存在t, 即发生路径冲突, 则在修改工件路径后重新执行寻找和移动操作. EFT策略的具体实现为: 通过将IM中第12和15行中的"$d+1$"替换为"执行EFT策略", 即可得到IIM(EFT策略见表2).

表 2 EFT策略

Table 2 EFT strategy

输入: $(N, M_h), S = s_1, s_2, \cdots, s_H, h$和$d = 0$
输出: d

1 计算M_h下的可行变迁集$\Delta(M_h)$; /*采用DAP计算M_h下的可行变迁*/
2 初始化$T_{\rm EFT}$和β为两个空列表;
3 for$(j = 1; j \leqslant |\Delta(M_h)|; j++)$
4　令t_j表示$\Delta(M_h)$第j个元素, $\beta_j = s_1, s_2, \cdots, s_{h-1}, t_j$;
5　$T_{\rm EFT}[j] = t_j, \beta[j] = \beta_j$;
6 end for
7 根据式(1)计算β中各变迁序列最后一个变迁的引发时间, 保留β中引发时间值最小的变迁序列并对应地更新$T_{\rm EFT}$;
8 if$(|T_{\rm EFT}| > 1)$
9　令$r \in$ rand$[1, |T_{\rm EFT}|]$;/*多个变迁具有相同的最小引发时间, 随机选1个*/
10 else
11　$r = 1$;/*具有最小引发时间的变迁只有1个*/
12 end if
13 for $(n = h; n \leqslant H; n++)$
14　if$(T_{\rm EFT}[r] = s_n)$
15　　$d = n - h$; break;
16　end if
17 end for
18 if$(d = 0)$/*$s_h, s_{h+1}, s_{h+2}, \cdots, s_H$中不存在安全变迁$T_{\rm EFT}[r]$,即产生了路径冲突*/
19　设$T_{\rm EFT}[r] = t_{i,l}, t_{i,l} \in T_i$, 设$\omega_i$和$\omega_{i'}$为同一类工件的加工路径;
20　for$(y = l; y \leqslant |T_i|; y++)$
21　　将$s_h, s_{h+1}, \cdots, s_H$中的第1个$t_{i',y}$替换为$t_{i,y}$;/*修改工件的加工路径*/
22　end for
23　for$(m = h; m \leqslant H; m++)$
24　　if $(T_{\rm EFT}[r] = s_m)$
25　　　$d = m - h$; break;
26　　end if
27　end for
28 end if
29 End

3.2 解码

对于一个可行的变迁序列S, 可以将其解码为一个由工件序号和工序组成的序列, 记作$\mathrm{JO}(S) = \mathrm{jo}_1, \mathrm{jo}_2, \cdots, \mathrm{jo}_H$, 其中$\mathrm{jo}_h = \mathrm{JN}_h[o_{i,l}]$, $\mathrm{JN}_h \in [1, \Phi]$为工件的序号, $o_{i,l}$为该工件的第l个工序. 解码的详细步骤如下, 记作算法3.

步骤 1　令$h = 1$, $\mathrm{JO}(S) = \mathrm{jo}_1, \mathrm{jo}_2, \cdots, \mathrm{jo}_H$.

步骤 2　将s_h解码为相应的工件序号和工序;

步 骤 2.1　如果$s_h \in \Theta_n$且$|\Omega_n| = 1$, 即s_h是J_n类工件VPT–路径上的变迁, 且J_n类工件只有一条VPT–路径, 若$s_h = t_{i,l}$是S中第y个$t_{i,l}$, 则$\mathrm{JN}_h = \sum_{a \in Z_{n-1}} \varphi_a + y$, $\mathrm{jo}_h = \mathrm{JN}_h[o_{i,l}]$; 否则转至步骤2.2.

步 骤 2.2　如果$s_h \in \Theta_n$且$|\Omega_n| > 1$, 即s_h是J_n类工件VPT–路径上的变迁, 且J_n类工件有多条VPT–路径;

步骤 2.2.1　若$s_h = t_{i,l}$, $|({}^{(p)}t_{i,l})^{\bullet}| > 1$, 且$^{\bullet}({}^{(p)}t_{i,l}) = \varnothing$, 即$s_h$是$J_n$类工件分支库所的输出变迁且分支库所为一个开始库所, 如果满足$t \in ({}^{(p)}t_{i,l})^{\bullet}$的变迁在$s_1, s_2, \cdots, s_{h-1}$中出现的次数为$y$, 则$\mathrm{JN}_h = \sum_{a \in Z_{n-1}} \varphi_a + y + 1$, $\mathrm{jo}_h = \mathrm{JN}_h[o_{i,l}]$; 否则转至步骤2.2.2;

步骤 2.2.2　若$s_h = t_{i,l}$, $|({}^{(p)}t_{i,l})^{\bullet}| > 1$, 且$^{\bullet}({}^{(p)}t_{i,l}) \neq \varnothing$, 即$s_h$是$J_n$类工件分支库所的输出变迁且分支库所不是开始库所, 如果满足$t \in ({}^{(p)}t_{i,l})^{\bullet}$的变迁在$s_1, s_2, \cdots, s_{h-1}$中出现的次数为$y$, 将$s_1, s_2, \cdots, s_{h-1}$中第$y+1$个满足$t \in {}^{\bullet}({}^{(p)}t_{i,l})$元素的位置记为$x$, 则$\mathrm{JN}_h = \mathrm{JN}_x$, $\mathrm{jo}_h = \mathrm{JN}_h[o_{i,l}]$; 否则转至步骤2.2.3;

步骤 2.2.3　若$s_h = t_{i,l}$, $|({}^{(p)}t_{i,l})^{\bullet}| = 1$, $|{}^{\bullet}({}^{(p)}t_{i,l})| = 1$, 且$s_h = t_{i,l}$是$S$中第$y$个$t_{i,l}$, 将$s_1, s_2, \cdots, s_{h-1}$中第$y$个$t = {}^{\bullet}({}^{(p)}t_{i,l})$的位置记为$x$, 则$\mathrm{JN}_h = \mathrm{JN}_x$, $\mathrm{jo}_h = \mathrm{JN}_h[o_{i,l}]$; 否则转至步骤2.2.4;

步骤 2.2.4　若$s_h = t_{i,l}$, $|({}^{(p)}t_{i,l})^{\bullet}| = 1$, $|{}^{\bullet}({}^{(p)}t_{i,l})| > 1$, 且$s_h = t_{i,l}$是$S$中第$y$个$t_{i,l}$, 将$s_1, s_2, \cdots, s_{h-1}$中第$y$个满足$t \in {}^{\bullet}({}^{(p)}t_{i,l})$的变迁位置记为$x$, 则$\mathrm{JN}_h = \mathrm{JN}_x$, $\mathrm{jo}_h = \mathrm{JN}_h[o_{i,l}]$.

步骤 3　$h = h + 1$, 若$h = H + 1$, 则输出$\mathrm{JO}(S)$; 否则返回步骤2.

例 2　考虑一个可行个体$S = t_{1,1}, t_{1,2}, t_{3,1}, t_{1,1}, t_{3,2}, t_{1,3}, t_{1,4}, t_{1,2}, t_{2,1}, t_{2,2}, t_{1,3}, t_{1,4}$. 根据算法3解码后得到的$\mathrm{JO}(S)$如图2所示.

3.3 模因组划分

以个体的完工时间为适应度值, 则可根据公式(1)计算种群中每个个体的适应度值, 记作$F(S)$, 并将种群个体按适应度值大小进行升序排序. 假设HDSFLA包含Meme个模因组, 则将排在第1(Meme + 1, 2Meme + 1, 3Meme + 1,等)的青蛙分配到第1个模因组, 排在第2(Meme + 2, 2Meme + 2, 3Meme + 2, 等)的青蛙分配到第2个模因组, 将排在第3(Meme + 3, 2Meme + 3, 3Meme + 3, 等)的青蛙分配到第3个模因组, 以此类推, 直至将所有青蛙划分完毕.

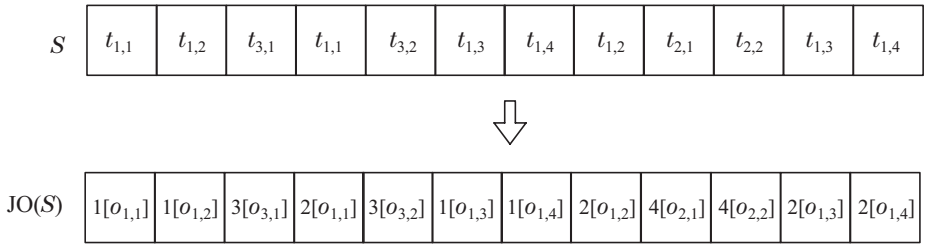


图 2 一个可行个体S及其解码序列JO(S)

Fig. 2 A feasible individual S and its JO(S)

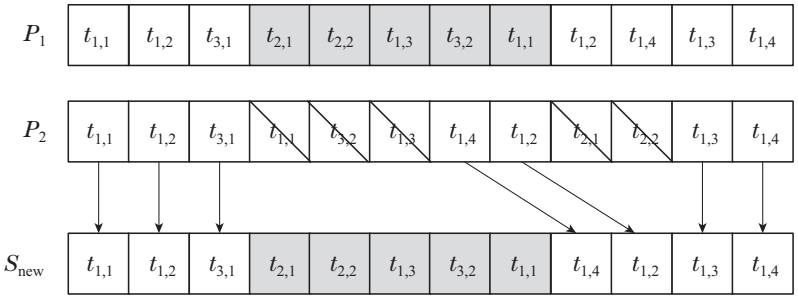
3.4 模因组内搜索

3.4.1 新个体生成

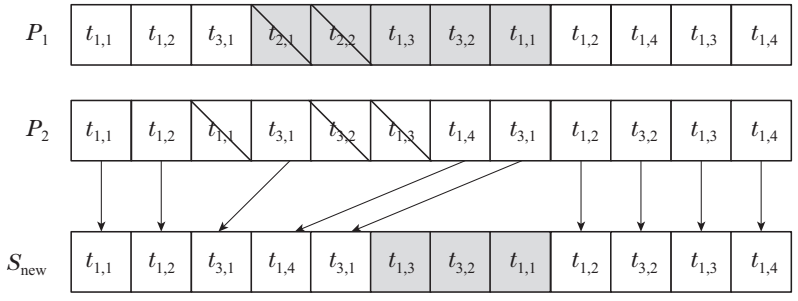
文中将采用两点交叉(two point crossover, TPX)操作来实现新个体的生成, TPX操作示意图如图3所示, 其中P_1和P_2为选定进行交叉的个体, $S_{\rm new}$为新生成的个体. 根据P_1 和P_2中工件的路径信息要考虑两种情况, TPX具体步骤如下:

步骤 1 随机生成h_1和h_2, h_1和h_2分别为交叉操作的起点和终点并满足 $h_1 \in (1, H)$, $h_2 \in (1, H)$ 和 $h_2 > h_1$; 本文令交叉段的长度在个体长度的1/6到1/3之间随机生成.

步骤 2 判定P_1中交叉段的元素是否在P_2中都有对应元素, 如果是, 则将P_1的交叉段复制到$S_{\rm new}$中相应位置, 把P_2中与交叉段相同的元素删除后, 将剩余的元素按先后顺序复制到$S_{\rm new}$的剩余位置, 见图3(a); 否则, 将P_1交叉段中满足对应关系的元素依次复制到$S_{\rm new}$中相应位置, 将P_2中相应元素删除后, 将剩余的元素按先后顺序复制到$S_{\rm new}$的剩余位置, 见图3(b).



(a) 交叉操作产生新个体(P_1和P_2的路径信息相同)



(b) 交叉操作产生新个体(P_1和P_2的路径信息不同)

图 3 交叉操作产生新个体

Fig. 3 New individual generated from P_1 and P_2

3.4.2 模因组内个体更新

对于每一个模因组, 模因组内的搜索执行N_M次. 在每次搜索中, 首先将模因组内的最优个体(记作$P_{\rm b}$)与最差个体(记作$P_{\rm w}$)进行交叉, 得到新个体$P_{\rm n1}$. 然后, 对$P_{\rm n1}$ 进行修正, 如果修正后$P_{\rm n1}$的适应度值优于$P_{\rm w}$, 则$P_{\rm n1}$替换$P_{\rm w}$; 否则, 将种群的最优个体(记作$P_{\rm g}$)与$P_{\rm w}$进行交叉, 得到新个体$P_{\rm n2}$. 如果修正后$P_{\rm n2}$的适应度值优于$P_{\rm w}$, 则$P_{\rm n2}$替换$P_{\rm w}$; 否则, 随机生成一个新个体, 修正使其可行并替换$P_{\rm w}$.

3.5 局部搜索

为了提升算法的局部搜索能力, 本文中采用了基于交换和插入算子的变邻域搜索方法 (variable neighborhood search, VNS)对种群中的最优个体进行改进, 基于交换算子的VNS, 记作S-VNS, 其执行步骤如下:

步骤 1 确定初始解S及其适应度值$F(S)$, 设$k = 0$, 重复步骤2–步骤4直到$k = {\rm LS}$;

步骤 2　在$(1, H)$内生成两个不同的随机数h_3和h_4, 交换S中第h_3个和第h_4个元素, 得到的新个体记作S', $k = k + 1$;

步骤 3　采用IIM修正S', 计算$F(S')$;

步骤 4　如果$F(S') < F(S)$, 则$S := S'$.

基于插入算子的VNS(记作I-VNS)的执行步骤与S-VNS类似, 唯一不同是在步骤2中将S中第h_4个元素插入到第h_3个元素之前. LS为S-VNS(I-VNS) 中执行交换(插入)操作的次数.

3.6　算法流程

HDSFLA流程见图4, 主要由种群初始化、解码、模因组划分、模因组内搜索及局部搜索5个模块构成.

4　实验仿真与分析

为了测试HDSFLA解决FAS调度问题的性能, 进行了算例测试和算法对比实验, 所有的测试算法均用Visual Studio C++ 2019编程实现, 运行环境为AMD Ryzen 7 4800HS CPU处理器, 主频2.9 GHz, 16 GB RAM的PC.

4.1　实验算例

由于没有可供参考的基准算例, 因此以文献[17, 24]中的制造装配系统(工件不具备柔性路径)作为参考, 并对其进行扩充, 得到如下一个FAS: 资源集为$R = \{m_1, m_2, \cdots, m_6\} \cup \{r_1, r_2, \cdots, r_6\}$, 工件集为$J = \{J_1, J_2, J_3\}$, 其中: $\{m_1, m_2, \cdots, m_6\}$为一组机器, 负责工件的加工; $\{r_1, r_2, \cdots, r_6\}$为一组机器人, 负责工件在相邻两个机器间的传送, 其对应的Petri网模型如图5所示. 分别在[5, 25]和[5, 10]之间随机生成机器和机器人上操作的加工时间, 记录在表3中.

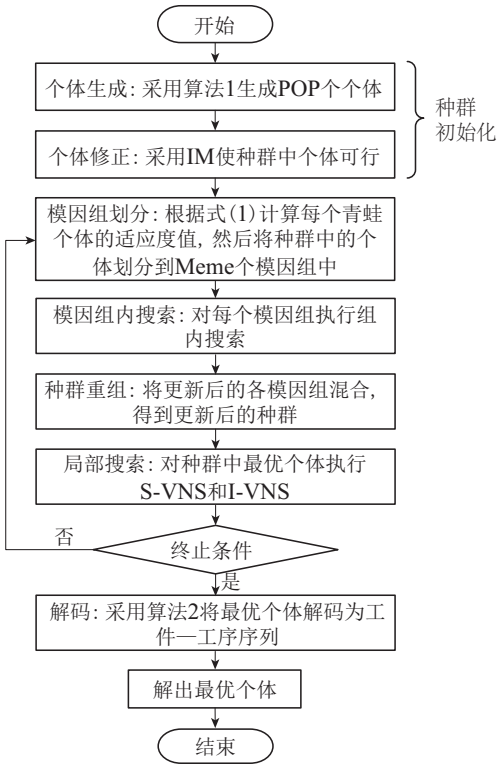


图 4　HDSFLA流程图

Fig. 4　Flowchart of HDSFLA

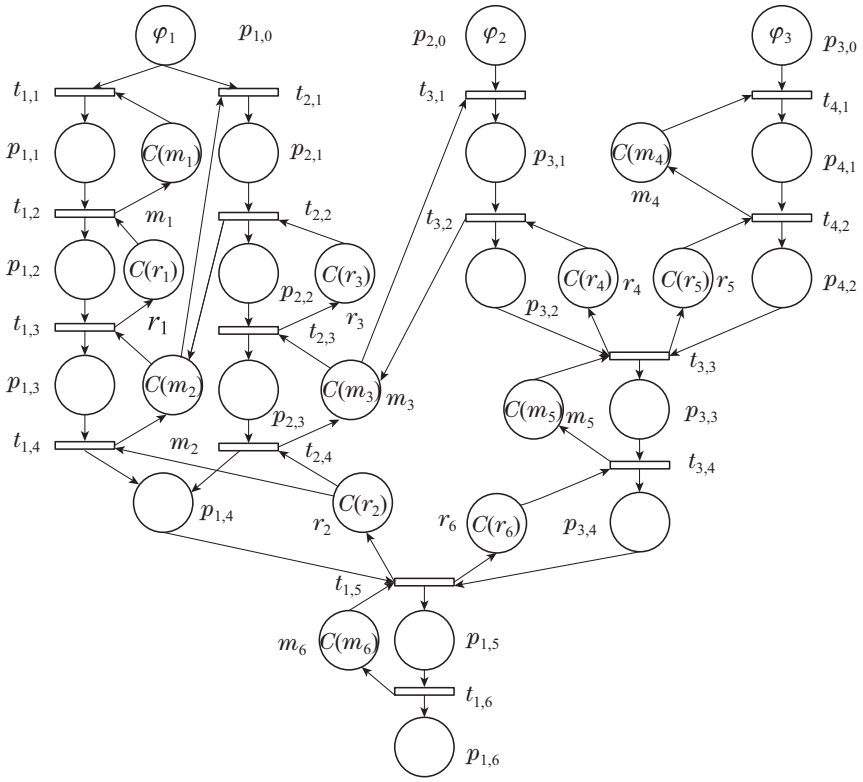


图 5　测试FAS的Petri网模型

Fig. 5　Petri net model of the test FAS

表 3 工件的加工时间

Table 3 Processing time of operations

J_1类工件		J_2类工件	J_3类工件
ω_1	ω_2	ω_3	ω_4
$d_{1,1}$: 11	$d_{2,1}$: 8	$d_{3,1}$: 18	$d_{4,1}$: 13
$d_{1,2}$: 6	$d_{2,2}$: 5	$d_{3,2}$: 8	$d_{4,2}$: 9
$d_{1,3}$: 17	$d_{2,3}$: 15	$d_{3,3}$: 20	
$d_{1,4}$: 7		$d_{3,4}$: 6	
$d_{1,5}$: 24			

基于该FAS生成20个具有不同工件数量和资源配置的测试算例, 记作FAS01-FAS20. 根据各算例中的资源配置将FAS01-FAS20划分为4组, 每组包含5个算例, 5 个算例中工件的个数$(\varphi_1, \varphi_2, \varphi_3)$分别为(20, 20, 20), (30, 30, 30), (50, 50, 50), (80, 80, 80)和(100, 100, 100). 每个算例中的资源数量分别为:

$$\text{FAS01}-\text{FAS05}: C(m_n)=1,\ C(r_n)=1,\ n\in Z_6;$$
$$\text{FAS06}-\text{FAS10}: C(m_n)=2,\ C(r_n)=2,\ n\in Z_6;$$
$$\text{FAS11}-\text{FAS15}: C(m_n)=3,\ C(r_n)=2,\ n\in Z_6;$$
$$\text{FAS16}-\text{FAS20}: C(m_n)=4,\ C(r_n)=2,\ n\in Z_6.$$

HDSFLA中有4个关键参数: 种群规模POP, 模因组个数Meme, 模因组内搜索次数N_M, 局部搜索中的搜索次数LS. 本文采用实验设计方法 (design of experiment, DOE) 来确定算法的参数设置. 首先, 给每个参数设置 3 个水平, 然后, 采用正交表$\mathrm{L}_9(4^3)$进行实验设置, 并用FAS09做测试算例. 参数的水平设置如表4所示(参数组合及对应结果见附录). 算法在每种参数组合下独立运行20 次, 20次运行的平均完工时间结果作为响应值 (response variable, RV), 基于各参数在不同水平下的平均RV, HDSFLA的参数设置为$\mathrm{POP}=60$, $\mathrm{Meme}=4$, $N_M=12$和$\mathrm{LS}=0.06H$.

表 4 算法参数水平设置

Table 4 Parameters and their factor levels

参数	参数水平		
	1	2	3
POP	60	80	100
Meme	4	5	10
N_M	8	10	12
LS	$0.04H$	$0.05H$	$0.06H$

4.2 算法有效性验证

对于每个算例, 每种算法的终止条件都设置为$T_{\mathrm{CPU}}=\Phi\times C_R\times 0.05$ s, 且算法单独运行20次. 算法在20次运行中的最好完工时间结果和平均完工时间结果分别记作BST和AVG(小数保留两位), 每个算例对应的最优结果标记为粗体.

4.2.1 算法各模块有效性验证

为了验证本文中提出的编码、IIM以及局部搜索模块的有效性, 本文测试了HDSFLA和其他3种DSFLA(表示为DSFLA_1, DSFLA_2和DSFLA_3)的性能. DSFLA_1是将HDSFLA中局部搜索的IIM改为IM获得的, DSFLA_2是将HDSFLA中的局部搜索模块除去后获得的, DSFLA_3是将DSFLA_2的编码改为现有文献[17,28]中的基于工件序号的编码方法后获得的. 为了保证比较的公正性, 不同DSFLA都采用相同的参数设置. 实验结果如表5所示.

表 5 HDSFLA和其他3种DSFLA的实验结果

Table 5 Experimental results of HDSFLA and 3 other DSFLA

算例	HDSFLA		DSFLA_1		DSFLA_2		DSFLA_3	
	BST	AVG	BST	AVG	BST	AVG	BST	AVG
FAS01	**532**	**532.00**	**532**	535.85	541	541.00	600	642.45
FAS02	**772**	**773.25**	**772**	775.15	781	781.00	939	995.80
FAS03	**1252**	**1252.80**	**1252**	1258.50	1261	1264.75	1470	1626.15
FAS04	**1972**	**1975.80**	1973	1981.65	1981	2018.95	2313	2594.80
FAS05	**2452**	**2456.50**	2454	2467.50	2466	2568.50	3021	3259.80
FAS06	**295**	**300.05**	300	313.15	321	338.05	530	564.30
FAS07	**412**	**421.75**	421	439.55	458	499.35	865	896.75
FAS08	**655**	**662.55**	666	682.20	739	814.15	1494	1566.50
FAS09	**1016**	**1028.85**	1029	1052.25	1077	1297.45	2514	2632.10
FAS10	**1257**	**1274.75**	1272	1300.15	1565	1646.35	3300	3359.25
FAS11	**220**	**222.15**	226	233.50	239	251.10	485	513.65
FAS12	**301**	**303.10**	305	316.30	334	359.60	773	818.65
FAS13	**460**	**468.50**	467	483.35	523	589.85	1435	1492.75
FAS14	**700**	**709.95**	711	744.60	814	983.25	2482	2529.10
FAS15	**868**	**877.30**	876	898.30	987	1219.95	3181	3251.05
FAS16	**181**	**183.65**	183	190.70	194	205.15	383	443.15
FAS17	**244**	**246.10**	247	256.55	277	298.85	733	763.65
FAS18	**364**	**369.00**	368	378.40	420	474.65	1364	1415.00
FAS19	**543**	**552.60**	552	569.60	622	794.25	2401	2457.60
FAS20	**663**	**679.50**	672	707.70	861	1040.85	3093	3163.85

从表5中可以看出, HDSFLA的表现优于其他3种DSFLA算法, 表明了HDSFLA在求解FAS调度问题时的有效性. 同时, HDSFLA 优于DSFLA_1的表现验证了IIM的有效性, HDSFLA优于DSFLA_2的结果验证了局部搜索模块的有效性, DSFLA_2远优于DSFLA_3的结果表明了本文提出的编码方式对于提升算法求解质量有着显著的效果. 此外, 算法性能还存在统计意义上的差异, 也说明了HDSFLA中不同模块的有效性, 实验结果和分析见附录.

4.2.2 HDSFLA与其他算法对比

由于FAS的生产特点以及死锁的存在, 目前无可用的调度算法能够解决其调度问题, 故选用几种尽可能相关的调度算法以及几种著名的算法作对比实验. 以HPSO[17], GA[28], GA_2, 灰狼算法(grey wolf optimizer, GWO)[29], 基于成功历史的自适应差分进化算法

(success-history based adaptive differential evolution, SHADE)[30]和jSO[31]为对比算法, 其中HPSO是用于求解不含柔性路径FAS调度问题的算法, GA用于求解含有柔性路径、不含装配操作的柔性制造系统调度问题, GA_2由GA改进得到, GWO, SHADE和jSO为近年来群智能和进化计算领域较为著名的算法. 由于这些算法无法直接用于解决FAS调度问题, 故做以下调整: 同时考虑文献[17, 28]中基于工件序号的编码方式, 修改为适用于FAS调度问题, 应用到HPSO, GA, GWO, SHADE和jSO中; 将文献[28]中的个体修复方法与文献[27]中的DAP结合来修正HPSO, GA, GWO, SHADE和jSO中的不可行个体. GA_2是将GA中的编码、个体修复、解码、交叉操作更换为本文提出的编码、个体修正、解码和交叉操作后得到. 比较算法的其他方面与来源参考文献保持一致. 实验结果如表6所示.

表 6 HDSFLA和6种比较算法的实验结果

Table 6 Experimental results of HDSFLA and 6 comparison algorithms

算例	HDSFLA		HPSO		GA		GA_2		GWO		SHADE		jSO	
	BST	AVG	BST	AVG	BST	AVG	BST	AVG	BST	AVG	BST	AVG	BST	AVG
FAS01	**532**	**532.00**	553	627.40	573	638.70	541	559.55	670	693.30	685	710.20	687	706.45
FAS02	**772**	**773.25**	886	967.75	847	968.55	786	815.25	1011	1058.20	1029	1062.20	1022	1065.35
FAS03	**1252**	**1252.80**	1473	1613.70	1458	1615.45	1288	1363.00	1715	1778.45	1725	1782.00	1748	1782.35
FAS04	**1972**	**1975.80**	2089	2593.80	2153	2449.45	1995	2189.15	2822	2868.30	2798	2871.85	2826	2869.30
FAS05	**2452**	**2456.50**	2893	3286.05	2793	3032.60	2561	2747.85	3542	3592.00	3551	3593.00	3521	3587.50
FAS06	**295**	**300.05**	319	334.65	431	486.60	354	378.35	378	433.25	567	595.10	546	579.75
FAS07	**412**	**421.75**	481	501.25	787	837.20	480	526.85	638	736.85	919	962.20	921	956.70
FAS08	**655**	**662.55**	901	998.70	1409	1515.35	773	823.60	1287	1391.10	1724	1782.75	1662	1754.65
FAS09	**1016**	**1028.85**	1658	1817.85	2488	2588.15	1132	1253.90	2281	2379.65	2939	3028.20	2936	3023.85
FAS10	**1257**	**1274.75**	2247	2384.35	3069	3303.70	1379	1535.45	2927	3047.35	3582	3836.10	3756	3851.75
FAS11	**220**	**222.15**	236	248.65	376	419.05	270	304.10	293	321.75	468	496.35	440	471.00
FAS12	**301**	**303.10**	354	384.15	651	730.75	388	428.30	500	544.80	802	844.95	754	805.35
FAS13	**460**	**468.50**	680	719.85	1286	1394.55	559	662.45	982	1087.65	1488	1580.50	1418	1513.85
FAS14	**700**	**709.95**	1270	1364.55	2330	2464.15	857	974.35	1844	1964.60	2745	2826.90	2624	2729.45
FAS15	**868**	**877.30**	1703	1933.60	3046	3190.25	1038	1165.35	2366	2510.15	3556	3662.70	3353	3608.85
FAS16	**181**	**183.65**	201	210.60	323	354.50	229	267.50	240	275.65	405	436.70	369	396.25
FAS17	**244**	**246.10**	299	314.95	556	641.95	346	393.80	398	474.70	695	756.50	650	693.35
FAS18	**364**	**369.00**	539	587.40	1181	1326.20	464	570.15	882	964.20	1356	1411.55	1254	1343.70
FAS19	**543**	**552.60**	1073	1146.55	2098	2362.95	699	835.50	1642	1741.65	2416	2557.20	2290	2412.10
FAS20	**663**	**679.50**	1372	1575.35	2824	3054.10	869	1007.25	2129	2238.95	3213	3374.60	3087	3253.25

由表6可知, HDSFLA在各算例上的BST和AVG结果均优于其他比较算法, 表明了HDSFLA在求解FAS调度问题时具有优异的性能. 同时, GA_2算法优于GA, 不仅表明了本文所设计的编码和交叉操作也适用于设计其他算法, 还表明了其有效性.

5 结论

本文研究了易死锁柔性装配系统的调度问题, 以完工时间为优化目标, 提出了HDSFLA. 首先, 建立了系统的Petri网模型; 然后, 基于Petri网模型建立了HDSFLA, 包括编码方法、个体修正方法、改进的个体修正方法、解码方法、生成新个体的交叉操作和局部搜索方法; 最后, 设计了对比实验验证了HDSFLA中所设计的不同模块的有效性, 与其他算法的对比表明了HDSFLA在求解易死锁柔性装配系统调度问题方面有着较强的优势.

鉴于针对柔性装配系统调度问题的研究尚处于初步阶段, 在后续研究中将继续关注以下几个方面: 1) 探索在优化能力和计算效率方面具有更好表现的求解算法, 如研究建立具有更低复杂度的个体修正方法、基于问题特征知识的搜索策略等; 2) 进一步关注具有其他生产约束的柔性装配系统, 如准备时间、不确定加工时间、零等待和随机事件等; 3) 考虑其他性能指标的优化, 如能耗、总提前/延迟时间、总流经时间等.

参考文献:

[1] FRAMINAN J M, GONZALEZ P P, VIAGAS V F. Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *European Journal of Operational Research*, 2019, 273(2): 401 – 417.

[2] KOMAKI G M, SHEIKH S, MALAKOOTI B. Flow shop scheduling problems with assembly operations: A review and new trends. *International Journal of Production Research*, 2020, 57(10): 2926 – 2955.

[3] CHEN Yaling, LEI Deming. An imperialist competition and cooperation algorithm for distributed two-stage assembly flow shop scheduling. *Control Theory & Applications*, 2021, 38(12): 1957 – 1967.
(陈雅玲, 雷德明. 求解分布式两阶段装配流水车间调度的帝国竞争—协作算法. 控制理论与应用, 2021, 38(12): 1957 – 1967.)

[4] WU C C, CHEN J Y, LIN W C, et al. A two-stage three-machine assembly flow shop scheduling with learning consideration to minimize the flowtime by six hybrids of particle swarm optimization. *Swarm and Evolutionary Computation*, 2018, 41: 97 – 110.

[5] KOMAKI G M, TEYMOURIAN E, KAYVANFAR V, et al. Improved discrete cuckoo optimization algorithm for the three-stage assembly flowshop scheduling problem. *Computers & Industrial Engineering*, 2017, 105: 158 – 173.

[6] ZHANG Z K, TANG Q H, CHICA M. Maintenance costs and makespan minimization for assembly permutation flow shop scheduling by considering preventive and corrective maintenance. *Journal of Manufacturing Systems*, 2021, 59: 549 – 564.

[7] SHAO Z S, SHAO W S, PI D C. Effective constructive heuristic and metaheuristic for the distributed assembly blocking flow-shop scheduling problem. *Applied Intelligence*, 2020, 50(12): 4647 – 4669.

[8] ZHAO F Q, SHAO D Q, WANG L, et al. An effective water wave optimization algorithm with problem-specific knowledge for the distributed assembly blocking flow-shop scheduling problem. *Knowledge-Based Systems*, 2022, 243: 108471.

[9] ZHAO F Q, DI S L, WANG L, et al. A self-learning hyper-heuristic for the distributed assembly blocking flow shop scheduling problem with total flowtime criterion. *Engineering Applications of Artificial Intelligence*, 2022, 116: 105418.

[10] ZHANG Z Q, HU R, QIAN B, et al. A matrix cube-based estimation of distribution algorithm for the energy-efficient distributed assembly permutation flow-shop scheduling problem. *Expert Systems With Applications*, 2022, 194: 116484.

[11] SHI F, ZHAO S K, MENG Y. Hybrid algorithm based on improved extended shifting bottleneck procedure and GA for assembly job shop scheduling problem. *International Journal of Production Research*, 2020, 58(9): 2604 – 2625.

[12] CHENG L X, TANG Q H, ZHANG L P, et al. Inventory and total completion time minimization for assembly job-shop scheduling considering material integrity and assembly sequential constraint. *Journal of Manufacturing Systems*, 2022, 65: 660 – 672.

[13] ZHANG S C, WANG S Y. Flexible assembly job-shop scheduling with sequence-dependent setup times and part sharing in a dynamic environment: Constraint programming model, mixed-integer programming model, and dispatching rules. *IEEE Transactions on Engineering Management*, 2018, 65(3): 487 – 504.

[14] WU X L, LIU X J, ZHAO N. An improved differential evolution algorithm for solving a distributed assembly flexible job shop scheduling problem. *Memetic Computing*, 2019, 11(4): 335 – 355.

[15] ZHANG S C, LI X, ZHANG B W, et al. Multi-objective optimisation in flexible assembly job shop scheduling using a distributed ant colony system. *European Journal of Operational Research*, 2020, 283(2): 441 – 460.

[16] XU J Y, NAGI R. Solving assembly scheduling problems with tree-structure precedence constraints: A lagrangian relaxation approach. *IEEE Transactions on Automation Science and Engineering*, 2013, 10(3): 757 – 771.

[17] LI X L, XING K Y, LU Q C. Hybrid particle swarm optimization algorithm for scheduling flexible assembly systems with blocking and deadlock constraints. *Engineering Applications of Artificial Intelligence*, 2021, 105: 104411.

[18] ZHANG G H, XING K Y, CAO F. Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion. *Engineering Applications of Artificial Intelligence*, 2018, 76: 96 – 107.

[19] MIYATA H H, NAGANO M S. The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Systems With Applications*, 2019, 137: 130 – 156.

[20] LIU S Q, KOZAN E, MASOUD M, et al. Job shop scheduling with a combination of four buffering constraints. *International Journal of Production Research*, 2018, 56(9): 3274 – 3293.

[21] LE C V, PANG C K. Robust total energy optimization of flexible manufacturing systems based on renyi mean-entropy criterion. *IEEE Transactions on Automation Science and Engineering*, 2016, 13(1): 355 – 367.

[22] MEJIA G, CABALLERO-VILLALOBOS J P, MONTOYA C. Petri nets and deadlock-free scheduling of open shop manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018, 48(6): 1017 – 1028.

[23] LI Z W, WU N Q, ZHOU M C. Deadlock control of automated manufacturing systems based on petri nets-a literature review. *IEEE Transactions on Systems, Man, and Cybernetics Part C-Applications and Reviews*, 2012, 42(4): 437 – 462.

[24] XING K Y, WANG F, ZHOU M C, et al. Deadlock characterization and control of flexible assembly systems with petri net. *Automatica*, 2018, 87: 358 – 364.

[25] EUSUFF M M, LANSEY K E, PASHA F. Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization. *Engineering Optimization*, 2006, 38(2): 129 – 154.

[26] MURATA T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 1989, 77(4): 541 – 580.

[27] LUO J C, LIU Z Q, ZHOU M C. A petri net-based deadlock avoidance policy for flexible manufacturing systems with assembly operations and multiple resource acquisition. *IEEE Transactions on Industrial Informatics*, 2019, 15(6): 3379 – 3387.

[28] LI X L, XING K Y, WU Y C, et al. Total energy consumption optimization via genetic algorithm in flexible manufacturing systems. *Computers & Industrial Engineering*, 2017, 104(C): 188 – 200.

[29] MIRJALILI S, MIRJALILI S M, LEWIS A. Grey wolf optimizer. *Advances In Engineering Software*, 2014, 69: 46 – 61.

[30] TANABE R, FUKUNAGA A. Success-history based parameter adaptation for differential evolution. *Proceedings of 2013 IEEE Congress on Evolutionary Computation*. Cancún: IEEE, 2013: 71 – 78.

[31] BREST J, MAUCEC M S, BOSKOVIC B. Single objective real-parameter optimization: Algorithm jSO. *Proceedings of 2017 IEEE Congress on Evolutionary Computation*. San Sebastian: IEEE, 2017: 1311 – 1318.

附录

附 件 https://pan.baidu.com/s/1QLIUPzCM3gJhK0mY9JhJkQ; 提取码: 0331

作者简介:

李晓玲 博士, 讲师, 目前研究方向为柔性制造系统控制与优化调度、智能仿真优化与调度等, E-mail: xiaolingli@chd.edu.cn;

冯彦翔 博士, 副教授, 目前研究方向为自动制造系统死锁控制与优化调度等, E-mail: fengyanxiang@xjtu.edu.cn;

张广辉 博士, 副教授, 目前研究方向为人工智能、分布式生产调度与控制等, E-mail: ghzhang@hebau.edu.cn;

段浩浩 硕士研究生, 目前研究方向为车间优化调度、车辆路径优化等, E-mail: 2357876237@qq.com.