文章编号: 1001-0920 (2014) 08-1345-09 DOI: 10.13195/j.kzyjc.2013.0847

# 柔性制造系统的改进粒子群无死锁调度算法

邢科义, 康苗苗, 郜振鑫

(西安交通大学 a. 系统工程研究所，b. 机械制造系统工程国家重点实验室，西安 710049)

**摘 要:** 基于柔性制造系统的 Petri 网模型, 以制造期最小为优化目标, 将死锁避免策略嵌入粒子群算法中, 提出一种无死锁改进粒子群调度算法. 该算法将粒子与工件的工序序列相对应, 以位置数值的大小表示对应工件工序在执行顺序中的优先级. 采用一步向前看的死锁避免策略方法对序列的可行性进行验证, 提出一种跳出局部极值的策略. 实例仿真结果表明了粒子群调度算法的可行性和有效性, 以及改进粒子群调度算法的优越性.

**关键词:** 制造系统；Petri 网；死锁避免策略；调度；粒子群算法

**中图分类号:** TP273 **文献标志码:** A

# Deadlock-free modified particle swarm optimization scheduling algorithm for flexible manufacturing systems

*XING Ke-yi*, *KANG Miao-miao*, *GAO Zhen-xin*

(a. Systems Engineering Institute，b. State Key Laboratory for Manufacturing Systems Engineering，Xi'an Jiaotong University，Xi'an 710049，China. Correspondent：XING Ke-yi，E-mail：kyxing@sei.xjtu.edu.cn)

**Abstract:** Based on Petri net models of flexible manufacturing systems and embedding the optimal deadlock avoidance policy into the particle swarm optimization(PSO) algorithm, a deadlock-free modified PSO scheduling algorithm is proposed to minimize the makespan. A particle is corresponded to a part sequence which is a permutation with repetition of parts. Real numbers in components of a position vector indicate priorities of the corresponding part operations. The feasibility of sequences is checked with the one-step look-ahead method in the optimal deadlock control policy, and a strategy for jumping out local optima is proposed. Experimental results show the effectiveness and feasibility of the particle swarm optimization for solving deadlock-free scheduling of flexible manufacturing systems and the modified particle swarm scheduling algorithm can perform much better.

**Key words:** manufacturing system；Petri nets；deadlock avoidance policy；scheduling；particle swarm optimization

## 0 引 言

柔性制造系统是由计算机控制的生产加工系统, 该系统中各种工件按预先规定的加工顺序进行操作, 工件对资源具有共享性. 当工件进入系统竞争有限资源时, 如果缺乏有效的分配方法, 则可能会出现死锁现象, 导致整个系统或部分系统无限期地阻塞, 无法完成加工任务. 因此, 有效避免死锁调度策略对于提高系统的性能有着重要的意义.

针对系统死锁问题, 人们从控制的角度进行了深入研究, 提出了多种死锁预防策略和死锁避免策略[1-5]. 然而, 这些控制策略仅能保证制造系统不会发生死锁, 但无法优化系统的运行性能, 因此制造系统的无死锁优化调度是一个值得深入研究的问题, 而目前关于这方面的研究相对较少[6-8]. 对于可能发生死锁的制造系统, 优化调度必须具备预防或避免死锁的机制. 本文将基于系统 Petri 网模型和死锁控制策略, 建立制造系统的无死锁优化调度. 要将一个死锁控制策略运用于优化调度算法中而不影响调度性能, 这样的控制策略必须具备计算简单和性能优越两个条件. 就笔者所知, 目前仅有两种死锁控制策略是具有多项式计算复杂性的最佳死锁控制策略: 一个是基于自动机[2], 另一个是基于系统 Petri 网模型[3].

制造系统无死锁优化调度问题不仅具有一般组合优化的特点, 更由于可能的系统死锁, 使得这一调度问题更加难以解决. Abdallaht 等[6]采用赋时 Petri

**收稿日期:** 2013-06-25；**修回日期:** 2013-11-28.

**基金项目:** 国家自然科学基金项目(50975224).

**作者简介:** 邢科义(1957–), 男, 教授, 博士生导师, 从事混合系统建模、控制和优化调度等研究；康苗苗(1987–), 女, 硕士生, 从事制造系统建模、控制与优化调度的研究.

网对制造系统建模, 提出了以制造期(makespan)最小为优化目标的无死锁调度算法, 通过搜索可达图来获得最优或次优的调度. Xu 等[7]基于制造系统的Petri网模型, 研究了系统的无死锁调度问题. 首先在假设工件缓存空间无限的条件下, 通过遗传算法求解调度问题; 然后分析在所得到的调度下运行实际系统时可能出现的死锁现象, 并通过增加必要的缓冲区来避免系统出现死锁. Xing 等[8]采用赋时Petri网对柔性制造系统建模, 以系统的制造期最短为优化目标, 将一种具有多项式复杂性的死锁避免策略(DAP)嵌入遗传算法中, 用DAP对遗传算法中染色体的可行性进行检测与修复, 以获得系统可行调度, 建立了一种无死锁遗传调度算法.

本文采用赋时Petri网对柔性制造系统建模, 研究以加工期最短为优化目标的无死锁调度问题, 建立柔性制造系统的无死锁粒子群调度算法. 本文利用工件序列表示工件的加工顺序, 工件在序列中的第$k$次出现代表工件的第$k$个加工处理. 把工件序列视为粒子的一部分, 将2行基本粒子矩阵扩展为3行矩阵(各行分别是工序序列、位置和速度), 并利用粒子位置值的大小表示对应工件工序加工的优先级, 同时约定数值越小优先级越高. 将粒子矩阵的列按照位置从小到大调整得到一个新粒子, 由于该新粒子的工件序列仅满足工件的工艺加工顺序要求, 但未必满足系统资源约束和控制约束, 本文提出一种集解码、检测、修复于一体的算法, 在对粒子的工序进行解码的同时, 对粒子进行安全性检测, 并给以合理的修复, 使得从任意一个粒子都可以得到一个可行的调度序列. 考虑到粒子群算法易于陷入局部极值, 受模拟退火算法中跳出局部极值思想的启发, 每一个粒子通过合适的规则产生一个新粒子, 如果新粒子比原粒子适应度高, 则接受该粒子, 否则以概率接受之. 实例仿真结果表明了本文提出的改进粒子群调度算法是可行且有效的, 其性能优于基本粒子群调度算法的性能.

## 1 Petri网基本定义及制造系统Petri网模型

### 1.1 Petri网的基本知识

Petri网[1]是一个三元组$N=(P,T,F)$. 其中: $P$是位置的集合, $T$是变迁的集合, $F\subseteq P\times T\bigcup(T\times P)$为有向弧集.

给定结点$x\in P\bigcup T$, ${}^\bullet x=\{y\in P\bigcup T|(y,x)\in F\}$表示$x$所有输入结点的集合, $x^\bullet=\{y\in P\bigcup T|(x,y)\in F\}$表示$x$的所有输出结点的集合.

$N$的标识或状态是一个映射$M:P\to \boldsymbol{Z}^+$, 其中$\boldsymbol{Z}^+=\{0,1,2,\cdots\}$. 给定位置$p\in P$和状态$M$, $M(p)$表示在$M$下$p$中标记token的个数. 变迁$t\in T$在标识$M$下使能当且仅当$\forall p\in {}^\bullet t, M(p)>0$, 记作$M[t>$; 在状态$M$下使能变迁$t$可以引发, 它的引发使系统达到新状态$M'$, 记作$M[t>M'$. 其中: $M'(p)=M(p)-1, \forall p\in {}^\bullet t\setminus t^\bullet$; $M'(p)=M(p)+1, \forall p\in t^\bullet\setminus {}^\bullet t$; 否则, $M'(p)=M(p)$. 对于一个变迁序列$\alpha=t_1t_2\cdots t_k$, 如果满足$M_i[t_i>M_{i+1}, i=1,2,\cdots,k$, 其中$M_1=M$, 则称变迁序列$\alpha$从状态$M$是可行的, 并称所有状态$M_i$是从$M$可达的. 把具有初始标识$M_0$的Petri网$N$记为$(N,M_0)$, 并用$R(N,M_0)$表示所有能从$M_0$可达的状态集合.

在Petri网$N$中, 一条路径是一个结点的序列$\alpha=x_1,x_2,\cdots,x_k$. 其中: $x_i\in P\bigcup T$, $(x_i,x_{i+1})\in F$, $i=1,2,\cdots,k-1$. 当$x_1=x_k$时, 路径$\alpha$是一条回路.

本文采用位置赋时Petri网来模拟系统的动态特征. 在位置赋时Petri网中, 标记必须在位置$p$至少停留一个给定的时间$d(p)$后才能离开.

### 1.2 制造系统及其Petri网建模

在本文中, 制造系统有$m$种资源, 资源集用$R=\{r_i,i=1,2,\cdots,m\}$表示, $r_i$的资源容量记为$C(r_i)$, 表示资源可以同时加工的最大工件数. 系统需加工$n$类工件, 工件类型集记为$J=\{J_1,J_2,\cdots,J_n\}$, 并用$\Psi(J_q)$表示要加工的$J_q$类工件数. 一条加工路径是一个操作序列, 工件可能有多条加工路径. $J_q$有$k(J_q)$条加工路径$w_1,w_2,\cdots,w_k$, 加工路径$w_i$是$l$个操作的序列, 即$w_i=O_{i1}O_{i2}\cdots O_{il}$. 其中: $O_{ij}$是该类工件在路径$w_i$中的第$j$个操作, $l$是路径$w_i$的长度. 本文假设每个操作需要一个资源, 相邻操作需要不同种类的资源, 因此一个加工路径与一个资源序列相对应, 于是$w_i$可以由资源序列$R(O_{i1})R(O_{i2})\cdots R(O_{il})$来确定.

本文采用文献[1, 3]给出的方法对制造系统进行建模, 工件的加工路径用Petri网中的位置和变迁组成的有向路径表示, 其中每个位置对应一个操作, 称为操作位置. 操作位置中的标记数表示该操作中正在加工处理的工件数. 一个变迁的引发对应一个操作的完成和下一个操作的开始. 为了方便, 对每一种工件引入两个虚拟的操作, 分别表示工件等待加工和工件已完成所有加工. 如$w_i$可写成$w_i=O_{qs}O_{i1}O_{i2}\cdots O_{il}O_{qe}$. 在Petri网中, $w_i$的模型为$\alpha_i=p_{qs}t_{i0}p_{i1}t_{i1}\cdots p_{il}t_{il}p_{qe}$, $p_{qs}$和$p_{qe}$分别对应虚拟操作, $p_{ij}$是模拟实际操作$O_{ij}$, $t_{ij}$是变迁. 则$J_q$型工件的加工路径集合可表示为

$$N_q=(P_q\bigcup\{p_{qs},p_{qe}\},T_q,F_q),\ J_q\in J,$$

其中$P_q$是所有实际操作对应的操作位置集合. 从$p_{qs}$到$p_{qe}$的有向路径对应$J_q$的一条加工路径. 在$N_q$中,

$\forall t \in T_q$, $|{}^\bullet t| = |t^\bullet| = 1$, 如果 $p \in P_q$, $|p^\bullet| > 1$, 则称 $p$ 为分支位置. 分支位置对应的操作完成后, 工件可以选择不同的加工路径, 即加工路径具有柔性.

在 Petri 网模型中, 每一类资源 $r$ 用一个位置表示, $r$ 中的标记数表示资源的容量, 因此 $r$ 的初始标记为 $C(r)$, 用 $P_R$ 表示所有资源位置的集合. 对于加工路径 $\alpha_i = p_{qs}t_{i0}p_{i1}t_{i1}p_{i2}t_{i2}\cdots p_{il}t_{il}p_{qe}$, 如果 $p_{ij}$ 对应的操作需要资源 $r$, 记为 $R(p_{ij}) = r$, 则添加从 $r$ 到 $t_{i(j-1)}$ 和从 $t_{ij}$ 到 $r$ 的有向弧表示资源需求和释放. 用 $F_R$ 表示与资源位置相关的有向弧集合, 则整个系统的 Petri 网调度模型 (PNS) 为

$$(N, M_0) = (P \bigcup P_s \bigcup P_f \bigcup P_R, T, F, M_0).$$

其中

$$P = \bigcup_{q\in J} P_q,\ P_s = \{p_{qs} | q \in J\},\ P_f = \{p_{qe} \mid q \in J\},$$

$$T = \bigcup_{q\in J} T_q,\ F = F_J \bigcup F_R,\ F_J = \bigcup_{q\in J} F_q.$$

初始标识 $M_0$ 定义为 $M_0(p_{qs}) = \Psi(J_q)$, $\forall p_{qs} \in P_s$; $M_0(p) = 0, \forall p \in P\bigcup P_f$; $M_0(r) = C(r), \forall r \in P_R$. 位置赋时 $d(p_{ij})$ 是 $p_{ij}$ 对应操作的加工时间, 这里没有给出, 将由调度问题确定.

用 ${}^{(o)}t$ 和 $t^{(o)}$ 分别表示变迁 $t$ 的输入和输出操作位置, ${}^{(r)}t$ 和 $t^{(r)}$ 分别表示变迁 $t$ 的输入和输出资源位置, 则 ${}^\bullet t = {}^{(o)}t \bigcup {}^{(r)}t$, $t^\bullet = t^{(o)} \bigcup t^{(r)}$. 给定标识 $M \in R(N, M_0)$, 如果 $M({}^{(o)}t) > 0$, 则称变迁 $t$ 在 $M$ 下是操作使能的; 如果 $M({}^{(r)}t) > 0$, 则称变迁 $t$ 在 $M$ 下是资源使能的.

**例 1**　以文献 [1, 3] 中的一个制造系统为例, 该系统由 4 台机器 $m_1 \sim m_4$ 和 3 个机器人 $r_1 \sim r_3$ 组成. 资源集为 $R = \{m_1, m_2, m_3, m_4, r_1, r_2, r_3\}$, $C(m_i) = 2$, $i = 1, 2, 3, 4$, $C(r_i) = 1$, $i = 1, 2, 3$. 系统加工 3 种工件 $J_1, J_2$ 和 $J_3$. $J_1$ 类工件的资源顺序是 $r_2m_2r_2$; $J_2$ 类工件有两条路径, 其资源顺序分别是 $r_1m_1r_2m_2r_3$ 和 $r_1m_3r_2m_4r_3$; $J_3$ 类工件的资源顺序是 $r_3m_4r_2m_3r_1$. 系统的 Petri 网模型如图 1 所示, 3 类工件需要加工的个数分别为 8、12 和 8. 其中: $P_s = \{p_{1s}, p_{2s}, p_{4s}\}$, $P_f = \{p_{1e}, p_{2e}, p_{4e}\}$, $P_R = \{m_1, m_2, m_3, m_4, r_1, r_2, r_3\}$, $P$ 为其余位置的集合.

当所有工件都加工完成时, 系统达到终止状态, 记作 $M_f$, $M_f(p) = M_0(p), \forall p \in P_R$; $M_f(p) = 0, \forall p \in P\bigcup P_s$; $M_f(p_{qe}) = M_0(p_{qs}), \forall p_{qe} \in P_f$.

一个调度即为一个可行变迁序列 $\alpha$, 满足 $M_0[\alpha > M_f$. 因此, 调度问题即是寻找一个可行变迁序列 $\alpha^*$, 满足 $M_0[\alpha^* > M_f$ 并且 $\alpha^*$ 中最后一个变迁的引发时间最小.

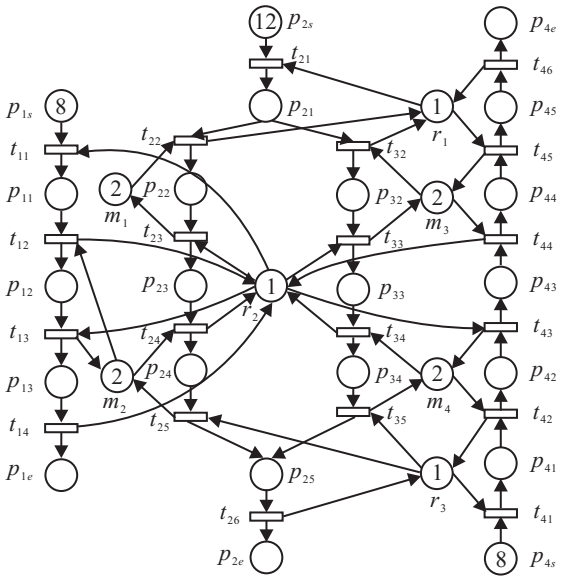


**图 1**　柔性制造系统的 Petri 网模型 (1)

## 2　死锁避免策略

本文的调度策略建立在死锁避免策略 (DAP) 的基础上. 由 PNS 和 S$^3$PR 模型的结构知, 两者的 DAP 是相同的. 本文将采用文献 [3] 中提出的具有多项式复杂性的 DAP, 将其嵌入粒子群算法中, 建立系统的无死锁改进粒子群调度算法. 为此, 简单回顾 DAP 的相关知识与结果.

### 2.1　死锁特征描述

文献 [3] 给出了 PNS 和 S$^3$PR 模型的 DAP 定义以及系统死锁的 Petri 网结构特征.

**定义 1**[3]　设 $(N, M_0)$ 是一个 PNS, $M \in R(N, M_0)$.

1) 标识 $M$ 是安全的, 当且仅当标识 $M_f$ 从 $M$ 是可达的, 否则 $M$ 是不安全的状态. 把从 $M_0$ 可达的安全 (不安全) 标识的集合记为 $S(N, M_0)(U(N, M_0))$.

2) 在安全标识 $M$ 下使能的变迁 $t$, 如果 $M[t > M'$, 并且 $M' \in S(N, M_0)$, 则称 $t$ 是安全的.

极大死锁避免控制策略 $\rho^*$ 通过阻止一些使能变迁的引发, 使得 PNS 的状态保持在安全可达标识集 $S(N, M_0)$ 内, 并且满足 $R(N, M_0, \rho^*) = S(N, M_0)$.

资源变迁回路 (RT- 回路) $\theta$ 是仅包含资源位置和变迁两类结点的有向回路. 用 $T[\theta]$ 和 $R[\theta]$ 分别表示 RT- 回路 $\theta$ 的变迁集合和资源位置集合. 如果满足 $({}^{(p)}T[\theta])^\bullet = T[\theta]$, 则称 RT- 回路 $\theta$ 是完备 RT- 回路 (PRT- 回路). 因为关于 $R_1$ 的两条 PRT- 回路的并仍为关于 $R_1$ 的 PRT- 回路, 故关于 $R_1$ 仅有一个极大 PRT- 回路 (MPRT-回路). 用 $\Omega$ 表示 Petri 网 $N$ 的所有 MPRT - 回路之集. 称 MPRT- 回路 $\theta$ 在状态 $M$ 下是饱和的, 当且仅当 $M({}^{(p)}T[\theta]) = \Psi(R[\theta])$. 因此, 系统是活的充要条件可表述如下.

**引理 1**[3] 一个 PNS $(N, M_0)$ 是活的 $\Leftrightarrow$ $(N, M_0)$ 的所有 MPRT- 回路在任何可达标识下均不会饱和.

设 $t$ 是 PNS $(N, M_0)$ 的一个变迁, $\theta$ 是一条 MPRT - 回路. 若 $t$ 的引发可增加 (减少) ${}^\bullet T[\theta]$ 中的标记数, 则称 $t$ 为 $\theta$ 的输入 (输出) 变迁. 分别用 $I(\theta)$ 和 $O(\theta)$ 表示 $\theta$ 的所有输入变迁和输出变迁的集合. 设 $\theta_1, \theta_2, \cdots, \theta_k$ 是一个 MPRT- 回路序列, $R_1 = \bigcap\limits_{i=1,2,\cdots,k} R(\theta_i) \neq \varnothing$, 如果资源 $r \in R_1$, $C(r) = 1$, 变迁 $t_i \in \theta_i \bigcap I(\theta_{i+1}), i = 1, 2, \cdots, k-1, k \geqslant 2$, 使得 ${}^{(r)}t_i = r$, 则称 $r$ 为中心资源.

### 2.2 不含中心资源系统的死锁避免策略

**引理 2**[3] 如果 PNS $(N, M_0)$ 不包含中心资源, 则它的所有不安全可达标识均为死锁标识.

引理 2 表明, 不含中心资源的 PNS 只包含安全和死锁两种状态. 因此, 为避免这类 PNS 中的死锁, DAP 只需阻止使系统从安全状态转移到死锁状态的变迁引发. 于是最优的 DAP 可通过一步向前看的方法获得, 即如果在安全标识下一个使能变迁引发后导致的新标识仍是安全的, 则该变迁是允许引发的. 新标识的安全性可以通过如下死锁检测算法[3]来判别.

死锁检测算法 (DS 算法).

令 $T^1$、$T^2$、$T^3$ 为变迁集, $R^1$ 为资源位置集.

输入: 安全标识 $M$ 以及在 $M$ 下的使能变迁 $t'$;

输出: $\chi(M, t')$; $/^*\chi(M, t') =$ 安全或不安全 $^*/$

初始化: $T^1 = T^2 = T^3 = R^1 = \varnothing$; $\chi(M, t') =$ 安全;

令 $r_1 = {}^{(r)}\ t'$; $M[t' > M_s$;

if $(M_s(r_1) \geqslant 1)$ {输出 $\chi(M, t')$; exit;} $/^* M_s$ 安全 $^*/$

else { $/^* M_s(r_1) = 0^*/$

$T^1 = \{t \in T \mid R({}^{(o)}t) = r_1, M_s({}^{(o)}t) \geqslant 1\}$;

$R^1 = \{r_1\}$ ;} //end if-else

while $(T^1 \setminus T^2 \neq \varnothing)$ do{

for $t \in T^1 \setminus T^2$, 令 $r = {}^{(r)}\ t$;

if $(M_s(r_1) \geqslant 1)$ {输出 $\chi(M, t')$; exit;}$/^* M_s$ 安全 $^*/$

else{ $/^* M_s = 0\ ^*/$

$T^3 := \{t \in T \mid R({}^{(o)}t) = r, M_s({}^{(o)}t) \geqslant 1\}$

$T^1 := T^1 \bigcup T^3$; $T^2 := T^2 \bigcup \{t\}$;

$R^1 := R^1 \bigcup \{r\}$; } //end if-else

} end while

$\chi(M, t') :=$ 不安全; 输出 $\chi(M, t')$; $/^* M_s$ 是死锁状态. 由 $T^1 \bigcup R^1$ 生成一个在状态 $M_s$ 下饱和的 RT- 回路, $T^1$ 中所有变迁在状态 $M_s$ 下都是死的. $^*/$

### 2.3 含中心资源系统的死锁避免策略

对于含中心资源的系统, 本文仍采用文献 [3] 中提出的先简化、后控制的方法, 即按照一定规则消除系统 Petri 网模型中的中心资源, 再对简化模型按照 DS 检测算法设计 DAP, 用简化模型对原系统进行监控得到次优的 DAP.

设 $r$ 是 PNS $(N, M_0)$ 的中心资源, $(N, M_0)$ 关于 $r$ 的简化模型记为 $(N(r), M_{A0})$, 可通过以下两步得到:

1) 从 $N$ 中删除 $r$ 及其相关弧, 记 $P_{R'} = P_R \setminus \{r\}$.

2) $\forall t \in P_{R'}^\bullet \bigcap {}^\bullet r, r_1 \in P_{R'}$, $(r_1, t) \in F$ 且 $(t, r) \in F$, 删除 $(r_1, t)$. 令 $p_s = {}^\bullet t$. $\forall t_s \in {}^\bullet p_s$, 如果 $(t_s, r_1) \notin F$, 则添加弧 $(r_1, t_s)$; 如果 $(t_s, r_1) \in F$, 则删除弧 $(t_s, r_1)$.

当 $(N(r), M_{A0})$ 不包含中心资源时, 将没有输入和输出资源位置的变迁的输入和输出操作位置视为一个位置. 故它有最佳的 DAP $\rho^*$ 且 $\rho^* || (N(r), M_{A0})$ 是活的 Petri 网. 由于 $N$ 和 $N(r)$ 有相同的变迁集合, 可采用 $\rho^* || (N(r), M_{A0})$ 对 $(N, M_0)$ 进行监督控制, 即使得一个变迁在 $(N, M_0)$ 中的使能引发等价于该变迁在 $\rho^* || (N(r), M_{A0})$ 中使能引发.

## 3 制造系统的无死锁调度粒子群算法

### 3.1 基本粒子群算法

粒子群优化算法 (PSO)[9]源于对鸟群捕食行为的研究, 它是一种进化算法, 从初始的随机解出发, 用适应度函数评价解的品质, 通过追随目前种群搜索到的最优值和个体最优值, 不断迭代更新以寻找全局最优解.

算法首先初始化一群随机粒子, 称为种群. 种群的大小为 $n$, 粒子根据一定的规则, 通过不断迭代找到最优解. 在迭代中, 每个粒子都需要更新自己的速度和位置, 主要通过追踪两个极值来更新: 一个是粒子本身所找到的最优解, 称为个体极值; 另一个是整个种群目前找到的最优解, 称为全局极值. 假设问题的解是 $D$ 维的, 则目标搜索空间是一个 $D$ 维的实数空间. 将第 $i$ 个粒子在 $k$ 时刻的位置和速度分别记作 $x_i[k] = (x_{i1}[k], \cdots, x_{iD}[k])$ 和 $v_i[k] = (v_{i1}[k], \cdots, v_{iD}[k])$, 则更新公式如下:

$$v_{id}[k+1] = wv_{id}[k] + c_1 r_{1id}[k](\mathrm{pbest}_{id}[k] - x_{id}[k]) + c_2 r_{2id}[k](\mathrm{gbest}_d[k] - x_{id}[k]), \tag{1}$$

$$x_i[k+1] = x_i[k] + v_i[k+1]. \tag{2}$$

其中: $r_{1id}[k]$ 和 $r_{2id}[k]$ 是均匀分布在 [0, 1] 之间的随机数; $c_1$ 和 $c_2$ 分别是自学习和社会学习因子, 合适的学习因子不但能加快收敛速度, 而且可以防止算法陷

入局部最优解, 通常情况下令两者都为 2; $\text{pbest}_i$ 是粒子 $i$ 的极值个体, gbest 是整个种群到目前为止找到的最优粒子; 惯性权重 $w = w_0 - \dfrac{w_0 - w_1}{p}k$, $w_0$ 和 $w_1$ 分别为惯性权重的初值和终值, $p$ 为最大迭代次数. 为了防止粒子速度过快而远离搜索空间, 或者粒子速度过小而陷入局部最优解, 粒子各维的速度应限制在一定范围内.

基本粒子群算法 (PSO).

1) 初始化种群. 设置种群大小, 最大迭代次数, 粒子的位置和速度, 个体极值和全局极值.

2) 评价粒子. 计算每个粒子的适应度函数值, 如果比该粒子当前的个体极值优, 则更新粒子的个体极值; 如果某个体极值比当前的全局极值优, 则更新种群的全局极值.

3) 利用式 (1) 和 (2) 更新粒子的速度和位置.

4) 判断是否结束. 如果迭代次数达到最大迭代次数, 则停止迭代并输出搜索到的最优解, 否则转到 2).

为了提高 PSO 算法的性能, 已出现多种改进粒子群算法[10-11], 这些算法不同程度地提高了粒子群算法的收敛速度和精度. 受模拟退火算法的启发, 本文提出一种跳出局部极值的策略, 以防止粒子群算法易陷入局部极值, 提高了算法的性能.

### 3.2　无死锁改进粒子群调度算法设计

采用 PSO 求解制造系统的无死锁调度, 需解决 3 个问题: 1) PSO 是为了解决连续问题提出的, 而调度问题是组合优化问题, 其解空间是离散的, 因此需要给出粒子与调度序列之间的对应关系; 2) 为了获得无死锁调度, 算法应具有避免死锁的机制, 本文将文献 [3] 中的 DAP 嵌入粒子群算法中, 以保证所得调度的无死锁性; 3) 如何改进粒子群算法使之能克服易陷入局部极值的缺陷. 本文受模拟退火算法思想的启发, 在引入惯性权重的同时, 提出一种跳出局部极值策略, 在每一次迭代过程中, 对每个粒子按照某种规则产生新粒子并以概率接受新粒子以提高解的质量.

#### 3.2.1　编　码

在粒子群算法中, 粒子是以位置和速度两个相关矢量的形式出现. 采用粒子群算法求解调度问题时首先需要建立调度问题的粒子表示.

将工件用正整数编号, 则工件集可记为 $J = \{1, 2, \cdots, N\}$, 其中 $N$ 为工件总数. 用 $O(J_i)$ 表示 $J_i$ 类工件加工路径的最大长度. 令 $D = \sum_{i=1}^{n}(O(J_i)+1)\cdot\Psi(J_q)$, $n$ 为工件种类的数目, 则调度问题的解空间是 $D$ 维的. 本文通过增加一个 $D$ 维工件序列向量 $\pi$, 将由位置和速度实数向量 $x$ 和 $v$ 形成的基本粒子群算法中粒子的 $2 \times D$ 矩阵表示扩展到 $3 \times D$ 矩阵表示, 即本文的一个粒子 $P$ 可以用一个 $3 \times D$ 矩阵来描述, $P = (\pi, x, v)^T$. 在 $\pi$ 中, 一个 $J_i$ 类工件 $q$ 出现 $O(J_i) + 1$ 次.

**例 2**　图 2 所示的系统可加工两类工件 $J_1$ 和 $J_2$, 设分别有 3 个和 2 个工件需要加工, 工件编码依次是 1, 2, 3, 4, 5. 工件加工路径长度都是 2, 故粒子的维数 $D = (2 + 1) \times 3 + (2 + 1) \times 2 = 15$, 各工件在粒子的工件序列向量 $\pi$ 中均出现 3 次. 若将粒子的位置和速度分别限制在 $[-5, 5]$ 与 $[-2.5, 2.5]$ 之间, 则一个粒子如表 1 所示.

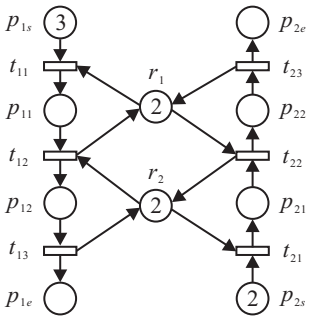


**图 2**　柔性制造系统的 Petri 网模型 (2)

**表 1**　一个粒子

| 工序 | 4 | 4 | 3 | 2 | 4 | 5 | 1 | 1 | 2 | 2 | 1 | 3 | 5 | 3 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 位置 | −1.8 | 0.8 | 4.4 | −2.1 | 2.1 | −2.4 | 1.4 | 3.4 | −4.3 | 2.9 | −0.5 | −3.5 | 4.9 | −0.1 | 4.1 |
| 速度 | −0.7 | −1.5 | −1.2 | 0.5 | −0.1 | −0.7 | 1.6 | 0.4 | 0.2 | 2.1 | −1.0 | 1.3 | 1.1 | −0.6 | −0.3 |

#### 3.2.2　解码、检测与修复

粒子的位置向量中数值的大小表示对应同一列上工序加工的优先级, 约定数值越小优先级越高. 因此首先将矩阵的列按照位置中数值从小到大调整, 为了方便, 将调整后的粒子仍记为 $P = (\pi, x, v)^{\mathrm{T}}$. 在 $\pi$ 中工件 $q$ 的第 $k$ 次出现表示该工件的第 $k$ 个操作开始加工, 最后一个 $q$ 表示该工件加工完成. 如果按每个工件都选择最长的一条路经加工, 则 $\pi$ 可解码为一个 Petri 网的变迁序列 $\alpha = t_0 \cdots t_{D-1}$. 注意此时 $\alpha$ 并不一定是可行的, 需要对其进行修正.

**例 3**　对表 1 所示的粒子, 将矩阵的列按照位置分量数值从小到大调整次序, 得到调整后的新粒子如表 2 所示. 其中: 工序中前 3 项 2、3、5 分别表示工件 2、3、5 的第 1 个操作开始加工, 由于工件加工路径

**表 2 调整后的粒子**

| 工序 | 2 | 3 | 5 | 2 | 4 | 1 | 3 | 4 | 1 | 4 | 2 | 1 | 5 | 3 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 位置 | −4.3 | −3.5 | −2.4 | −2.1 | −1.8 | −0.5 | −0.1 | 0.8 | 1.4 | 2.1 | 2.9 | 3.4 | 4.1 | 4.4 | 4.9 |
| 速度 | 0.2 | 1.3 | −0.7 | 0.5 | −0.7 | −1.0 | −0.6 | −1.5 | 1.6 | −0.1 | 2.1 | 0.4 | 0.3 | −1.2 | 1.1 |

唯一确定且工件2和3具有同一加工路径, 前3项对应的变迁分别为$t_{11}, t_{11}, t_{21}$; 第4项2表示工件2的第2个操作开始加工, 对应变迁$t_{12}$; 以此类推, 可将粒子解码为对应的变迁序列$\alpha = t_{11}t_{11}t_{21}t_{12}t_{21}t_{11}t_{12}t_{22}t_{12}t_{23}t_{13}t_{13}t_{22}t_{13}t_{23}$. 这里$\alpha$不可行, 因为$M_0[t_{11}t_{11}t_{21}t_{12} > M_1 = (1,1,1,0,1,1,0,0,1,0)$, 在$M_1$下是$t_{21}$非资源使能的, 故不能引发.

由于直接解码后的变迁序列不一定可行, 本文在对粒子进行解码的同时进行可行性与安全性检查, 并作合理的调整, 以使得到的变迁序列是安全、可行的. 于是整个过程可用如下的算法DCA来描述. 在这个过程中粒子的位置和速度数值保持不变.

算法DCA (Decoding, checking, and amending).

输入: 粒子$P = (\pi, x, v)^{\mathrm{T}}$, 初始标识$M_0$;

输出: 修正后的粒子$P$及其对应的可行变迁序列$\alpha$.

Begin

设$\pi = s_0 s_1 \cdots s_{D-1}$; $\alpha := \varepsilon$; //$\varepsilon$为空串.

for $(i = 0; i < D; i++)$ do {

for $(j = 0; j < D - i; j++)$ {

令$A = \varnothing$; $q = \pi[i + j]$; //$q$表示工件.

设$q$在位置$p$中, 令$E = \{t \in p^{\bullet} \mid t$在$M_i$下使能$\}$;

while $(E \backslash A \neq \varnothing)${ //寻找可行路径.

在$E \backslash A$中任意选择一个变迁$t$;

if $(\chi(M_i, t) =$安全$)$ { //判断$t$在$M_i$下的安全性.

令$t_i = t$; $M_i[t_i > M_{i+1}$; $\alpha := \alpha t_i$;

$\pi := s_0 s_1 \cdots s_{i-1} s_{i+j} s_i \cdots s_{i+j-1} s_{i+j+1} \cdots s_{D-1}$;

break; } //工件选出可行路径, 跳出while循环.

else $A := A \bigcup \{t\}$; //$t$在$M_i$下非安全, 故非控制使能.

} end while

if $(E \neq A)$ break; /*$E \neq A$表明在$M_i$下, 工件$q = \pi[i + j]$能安全地进行下一步加工并已选到可行路径, 故跳出for $(j = 0; \cdots)$循环. 因状态$M_i$是安全的, 故一定存在$\pi[i + j]$以后的一个工件, 它在$M_i$下能安全地进行下一步加工 */

} //end for $(j \cdots)$

} //end for $(i \cdots)$

输出粒子$P$及其对应的可行变迁序列$\alpha$.

End

注意, 跳出while循环有两种情况: 一是找到路径, 从break跳出while, 此时$E \neq A$; 另一种是while在$E = A$时结束, 此时没有找到可行路径, 即工件$q$不能开始下一个操作.

算法在对粒子进行解码时, 也进行安全性检查和合理的修复. 在当前状态$M_i$下, 若能从工件$q = \pi[i]$的下一步多条可选加工路径中随机地选择一条可行加工路径(即对应的变迁$t$是安全的), 则开始工件$q$的下一步加工, 将工件$q = \pi[i]$解码为变迁$t_i$; 否则从$q = \pi[i]$之后, 逐一判断工件$\pi[i + 1], \pi[i + 2], \cdots$能否开始下一步的加工. 由于$M_i$是安全的, 一定存在$j$使得$\pi[i + j]$在$M_i$下的下一步加工能够开始并在粒子$P$中将$\pi[i+j]$对应的列移动到第$i$列. 再对粒子的第$i+1$列对应的工件重复以上解码、检测和修复. 最终得到一个可行的变迁序列, 即一个无死锁调度.

**例4** 按上述方法对表2所示粒子$P = (\pi, x, v)^{\mathrm{T}}$进行调整解码, 得到新粒子$P' = (\pi', x, v)^{\mathrm{T}}$. 其中: 新的工序序列$\pi' =$(2, 3, 5, 2, 1, 2, 3, 5, 4, 3, 1, 4, 4, 1, 5), 修复前后粒子的速度和位置向量$x$、$v$不变.

### 3.2.3 适应度函数

本文以粒子对应调度的制造期makespan作为适应度函数$F$. 适应度函数值越小, 粒子对应调度的性能越好. 为计算一个粒子$P$的适应度函数值$F(P)$, 首先利用算法DCA将粒子解码为对应的可行变迁序列$\alpha = t_0 t_1 t_2 \cdots t_{D-1}$. 用$t_k[O_{ij}]$表示在解码过程中变迁$t_k$对应第$i$类工件的第$j$个操作, $f(t_k[O_{ij}])$表示变迁$t_k$的引发时间. 则$t_k[O_{ij}]$必须在操作$O_{i(j-1)}$加工完之后引发以满足工件的加工顺序, 同时在$t_{k-1}$引发之后引发. 假设$t_{k-1}$对应操作$O_{uv}$, $t_s$对应操作$O_{i(j-1)}$, 从而有$f(t_k[O_{ij}]) = \max\{f(t_s[O_{i(j-1)}]) + d(O_{i(j-1)}), f(t_{k-1}[O_{uv}])\}$, 而且粒子$P$对应的适应度函数值$F(P) = \mathrm{makespan}(P) = f(t_{D-1})$.

**例5** 考虑图2所示的系统及表2所示的粒子. 由例4知, 修复后粒子的工序序列$\pi' =$(2, 3, 5, 2, 1, 2, 3, 5, 4, 3, 1, 4, 4, 1, 5), 其对应的变迁序列$\alpha = s_0 s_1 s_2 \cdots s_{14} = t_{11}t_{11}t_{21}t_{12}t_{11}t_{13}t_{12}t_{22}t_{21}t_{13}t_{12}t_{22}t_{23}t_{13}t_{23}$. 设各工件的第1、第2个操作所用时间分别为2和4, 即$d(O_{11}) = d(O_{21}) = 2$, $d(O_{12}) = d(O_{22}) = 4$. 设系

统开始时间为 0. 则有

$$f(s_0[O_{11}])=0;$$

$$f(s_1[O_{11}])=\max\{0,f(s_0[O_{11}])\}=0;$$

$$f(s_2[O_{11}])=\max\{0,f(s_1[O_{11}])\}=0;$$

$$f(s_3[O_{12}])=\max\{f(s_0[O_{11}])+d(O_{11}),f(s_2[O_{11}])\}=\max\{0+2,0\}=2;$$

$$f(s_4[O_{11}])=\max\{0,f(s_3[O_{12}])\}=\max\{0,2\}=2;$$

$$f(s_5[O_{1e}])=\max\{f(s_3[O_{12}])+d(O_{12}),f(s_4[O_{11}])\}=\max\{2+4,2\}=6;$$

其余计算以此类推.

#### 3.2.4　粒子更新

给定粒子 $P$, 粒子更新就是对粒子的位置和速度矢量进行更新. 设粒子 $P$ 的第 $d$ 列 $(d=1,2,\cdots,D)$ 代表工件 $q$ 的第 $m$ 次操作, 而这一操作分别是在取得个体极值粒子的第 $b$ 列和取得全局极值粒子的第 $c$ 列, 则速度按下式更新:

$$\begin{aligned}&v_{id}[k+1]=\\&wv_{id}[k]+c_1r_{1id}[k](\mathrm{pbest}_{ib}[k]-x_{id}[k])+\\&c_2r_{2id}[k](\mathrm{gbest}_c[k]-x_{id}[k]),\end{aligned}\tag{3}$$

而位置则按式 (2) 更新.

#### 3.2.5　跳出局部极值策略

由于粒子群算法易陷入局部极值, 为避免算法过早陷入局部极值, 受模拟退火算法思想的启发, 本文提出一种跳出局部极值的策略. 在该策略中, 对每个粒子按照一定的规则生成新粒子, 然后以概率接受这个粒子. 这里有两个问题需要考虑: 一是如何生成新粒子; 二是如何选择概率接受函数.

1) 生成新粒子的方法. 从给定粒子 $P$ 产生新粒子 $P'$, 可按以下方法进行. 以 $P$ 的工序为父代工序 $\pi_1$, 取随机数 $r=\mathrm{random}[0,1]$, 如果 $r\in[0,0.6)$, 则选择全局极值粒子的工序作为交叉段的供体 $\pi_2$; 否则, 选择个体极值粒子的工序作为供体 $\pi_2$. 在 $\pi_2$ 中随机选择交叉段 $\sigma$. $\sigma$ 的起点 $A$ 将父代和供体均分为两部分, 取交叉段长度 $L$ 在第 2 部分长度的 1/3 到 1/2 之间. 对选定交叉段 $\sigma$ 中的每个工件 $q$, 如果 $q$ 在 $\pi_2$ 中是第 $k$ 次出现, 则将 $\pi_1$ 中第 $k$ 次出现的 $q$ 改写为 X, 得到 $\pi_3$. 将 $\sigma$ 插入到 $\pi_3$ 的 $A$ 点与 $A+1$ 点之间, 再将所有 X 位删除, 得到 $\pi$. 将 $\pi$ 和粒子 $P$ 的位置与速度相结合得到新粒子 $P'$.

**例 6**　给定粒子 $P$ 的工序序列 $\pi_1=(1,1,1,2,2,2,3,3,3,4,4,4,5,5,5)$. 假设以 $\pi_2=(2,3,5,2,1,2,3,5,4,3,1,4,4,1,5)$ 作为供体. 选择 $A=4,L=5$, 则在 $\pi_2$ 上的交叉段 $\sigma=(1,2,3,5,4)$. $\sigma$ 中的工件 1、2、3、5、4 在 $\pi_2$ 上分别是第 1、3、2、2、1 次出现, 将 $\pi_1$ 中对应这些工件的位置改为 X, 得 $\pi_3=(\mathrm{X},1,1,2,2,\mathrm{X},3,\mathrm{X},3,\mathrm{X},4,4,5,\mathrm{X},5)$, 将 $\sigma$ 段插入 $\pi_3$ 的 $A$ 点, 并删除所有 X 位, 得到新粒子 $P'$ 的工序序列 $\pi=(1,1,2,1,2,3,5,4,2,3,3,4,4,5,5)$.

2) 概率接受函数的选取. 给定原粒子 $P$、生成的新粒子 $P'$ 以及全局极值粒子 $P_{\mathrm{gbest}}$. 令 $\Delta=F(P')-F(P)$, $\Delta'=F(P)-F(P_{\mathrm{gbest}})$. 若 $\Delta'<0$, 则粒子 $P$ 比全局极值粒子 $P_{\mathrm{gbest}}$ 优, 更新全局极值粒子; 若 $\Delta'\geqslant 0$, 则以 $\min\{1,\exp[-\Delta\times\Delta']\}$ 接受 $P'$, 即当 $\Delta>0$ 时 (此时 $0<\min\{1,\exp[-\Delta\times\Delta']\}<1$) 且 $\mathrm{random}[0,1]\leqslant\min\{1,\exp[-\Delta\times\Delta']\}$, 接受新粒子 $P'$; 而当 $\Delta\leqslant 0$ 时 (此时 $\min\{1,\exp[-\Delta\times\Delta']\}=1$), 则接受新粒子. 总之, 若新粒子 $P'$ 的性能较原粒子 $P$ 优, 则接受 $P'$; 若 $P'$ 的性能较 $P$ 差, 则以 $\mathrm{random}[0,1]\leqslant\min\{1,\exp[-\Delta\times\Delta']\}$ 接受 $P'$. 由指数函数的性质知: $P'$ 的性能越优, $\Delta$ 越小, 接受的概率就越大; 反之接受的概率就小. 由此可知该概率公式选择的合理性. 结合粒子群算法的优化过程可以改善粒子群算法易陷入局部极值的缺点.

本文的跳出局部极值策略算法可以描述如下.

算法 SGO.

输入: 粒子 $P$ 及其个体极值 $P_{\mathrm{lbest}}$, 全局极值 $P_{\mathrm{gbest}}$;

if $(\Delta'=F(P)-F(P_{\mathrm{gbest}})>0)\{$

按 1) 中提出的方法生成新粒子 $P'$;

计算 $F(P')$ 及 $\Delta=F(P')-F(P)$;

if $(\mathrm{random}[0,1]\leqslant\min\{1,\exp[-\Delta\times\Delta']\})$

更新粒子 $P$, $P\leftarrow P'$;$\}$

输出: 新粒子 $P$.

#### 3.2.6　无死锁调度问题的改进粒子群算法

本文的粒子是一个 $3\times D$ 矩阵, 3 行分别为工序序列、位置向量和速度向量. 各位置的值表示对应工件加工的优先级, 其值越小, 优先级越高. 将粒子的各列按位置值从小到大的顺序排列, 可解码成一个对应的变迁序列, 而这一变迁序列常常是不可行的, 对不可行变迁序列或粒子需要进行修复.

本文将解码、检测与修复融为一体, 通过解码、检测与修复算法 DCA 将不可行粒子修复为可行粒子, 并得到对应的可行变迁引发序列, 即调度方案.

在改进粒子群算法中, 每个粒子分别通过式 (3) 和 (2) 来调整自己的速度和位置, 适应度函数值取为工件加工完成时间, 通过粒子对应的调度序列来求得. 为了缓解粒子群算法易于陷入局部极值的问题, 本文提出一种跳出局部极值的策略, 以某概率保留个体极

值和全局极值的优秀片段. 整个改进粒子群无死锁调度算法可表述如下.

改进粒子群无死锁调度算法 (MPSO).

Step 1: 初始化种群中粒子, 学习因子 $c_1$ 和 $c_2$, 惯性权重 $w$, 最大迭代次数 gen_max, $\text{gen} = 0$.

Step 2: 对每个粒子 $P_i$ 执行如下操作:

按照位置值从小到大的顺序调整粒子矩阵;

利用算法 DCA 对 $P_i$ 进行解码、检测和修复;

计算 makespan $(P_i)$;

SGO$(P_i)$; /* 将跳出局部极值策略应用于 $P_i$ */

更新 $P_i$ 的个体极值 $P_{\text{ibest}}$ 和全局极值 $P_{\text{gbest}}$.

Step 3: 根据式 (3) 和 (2) 更新每个粒子的速度和位置, 更新惯性权重 $w$, $\text{gen} = \text{gen} + 1$.

Step 4: 判断是否结束, 若 $\text{gen} < \text{gen\_max}$, 则转向 Step 2.

Step 5: 输出全局最好粒子及其对应的引发变迁序列.

由于 DS 算法是多项式的, 嵌入 DS 算法本质上不会改变粒子群算法的复杂性, 即如果粒子群算法的复杂性是多项式的, 则嵌入 DS 算法后其复杂性仍为多项式的; 算法 DCA 在对粒子解码的同时进行了安全检测, 将不安全的序列转换为安全的, 同时给出可行调度序列. 因此由本算法得到的调度序列是无死锁的.

## 4 实例仿真与结果分析

本节针对图 1 所示的制造系统的调度问题, 分别采用基本粒子群算法 PSO 和 MPSO 进行仿真求解. 由文献 [3] 知, 系统中有 18 个 MPRT- 回路可能导致系统死锁. 考虑 $C(r_2) = 1, 2$ 两种情形. 若 $C(r_2) \geqslant 2$, 则系统不含中心资源, 可直接采用死锁检测算法 DS 作为 DAP; 若 $C(r_2) = 1$, 则 $r_2$ 为唯一的中心资源, 简化后的系统模型中不含中心资源, 有 4 个 MPRT- 回路. 可将死锁检测算法 DS 应用到化简后的模型. 为了便于比较, 采用文献 [8] 中给出的加工时间数据, 加工时间如表 3 所示.

**表 3 图 1 所示系统的工件加工时间**

| $J_1$ | $J_2$ | | $J_3$ |
|---|---|---|---|
| $w_1$ | $w_2$ | $w_3$ | $w_4$ |
| $O_{11}$: 8 | $O_{21}$: 4 | $O_{21}$: 4 | $O_{31}$: 5 |
| $O_{12}$: 34 | $O_{22}$: 32 | $O_{22}$: 23 | $O_{32}$: 22 |
| $O_{13}$: 5 | $O_{23}$: 8 | $O_{23}$: 6 | $O_{33}$: 4 |
| — | $O_{24}$: 38 | $O_{24}$: 20 | $O_{34}$: 17 |
| — | $O_{25}$: 5 | $O_{25}$: 5 | $O_{35}$: 6 |

本文的仿真平台是基于 1.19 GHz 的 Inter Pentium PC 机, VC++6.0. 仿真参数设置为: 迭代次数为 10 000, 种群包含 20 个粒子, 学习因子 $c_1 = 2, c_2 = 2$, 考虑不含惯性权重和惯性权重选取 3 组值, 分别是 $1.0 \sim 0.6$, $0.9 \sim 0.6$, $0.9 \sim 0.5$, 粒子的速度在整个过程中是 $[-5, 5]$ 之间的随机实数, 位置的初值是 $[-10, 10]$ 之间的随机实数, 运行 10 次并记录其平均 makespan 以及最优的 makespan.

**情形 1** $C(r_2) = 2$, 此时系统不含中心资源, 仿真结果如表 4 所示.

**表 4 不含中心资源系统运行结果**

| $w_0 \sim w_1$ | PSO | | MPSO | |
|---|---|---|---|---|
| | Average | Best | Average | Best |
| $1.0 \sim 1.0$ | 361.3 | 341 | 340.8 | 336 |
| $1.0 \sim 0.6$ | 347.2 | 315 | 328.5 | 310 |
| $0.9 \sim 0.6$ | 349.9 | 335 | 339.0 | 325 |
| $0.9 \sim 0.5$ | 353.5 | 330 | 330.2 | 307 |

10 次仿真给出的最好调度方案是 MPSO 在惯性权重取值范围为 $0.9 \sim 0.5$ 时得到的, 其 $\text{makespan} = 307$.

图 3 给出了以惯性权重取值范围为 $0.9 \sim 0.5$ 的运行过程中全局极值 makespan 的变化过程, 其中虚线和实线分别代表 PSO 和 MPSO 的运行情况. PSO 的最好结果为 $\text{makespan} = 330$, MPSO 的最好结果为 $\text{makespan} = 307$.

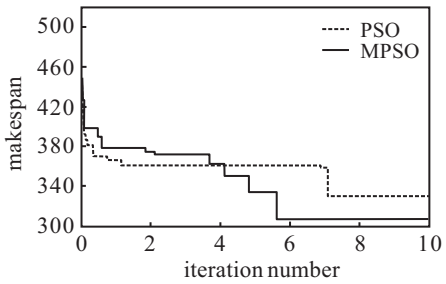


**图 3 两种算法最好解的对比**

**情形 2** $C(r_2) = 1$, 系统包含中心资源 $C(r_2)$, 仿真结果如表 5 所示. 10 次仿真给出的最好调度方案是 MPSO 在惯性权重取值范围为 $1.0 \sim 0.6$ 时得到的, 其 $\text{makespan} = 387$.

**表 5 含中心资源系统运行结果**

| $w_0 \sim w_1$ | PSO | | MPSO | |
|---|---|---|---|---|
| | Average | Best | Average | Best |
| $1.0 \sim 1.0$ | 446.2 | 429 | 416.2 | 402 |
| $1.0 \sim 0.6$ | 433.5 | 421 | 407.6 | 387 |
| $0.9 \sim 0.6$ | 434.4 | 419 | 408.7 | 391 |
| $0.9 \sim 0.5$ | 424.9 | 408 | 405.5 | 392 |

结果分析如下. 表 4 和表 5 给出了 $C(r_2) = 2$ 和 1 两种情况下的仿真结果, 第 1 行的结果是未加惯性权重. 从纵向比较可知, 在 PSO 和 MPSO 中加入惯性权重均可改善算法的性能; 从横向比较可知, 在粒子群算法中引入跳出局部极值策略, 改善了算法的性

能, 即 MPSO 比 PSO 的性能更优; 由于 MPSO 肯定接受好的新粒子, 而以某概率接受较差的粒子, PSO 中没有接受差的粒子这个操作. 因此就最好值而言, 在目标寻优过程初期, PSO 可能比 MPSO 表现较优; 随着迭代的进行, 较差粒子接受的概率逐渐减小, MPSO 逐渐体现出跳出局部极值的优点. 图 3 的结果和理论分析是一致的. 因为本文结合了死锁控制策略, 将不可行的序列通过死锁检测与修复转换为可行的序列, 所以得出的调度序列是无死锁的, 并且通过软件 Pipe 3.0 在线验证了调度序列的无死锁性. 文献 [8] 给出的含与不含中心资源两种情形下的最好调度 makespan 为 447 和 321. 本文给出的最好调度方案对应的 makespan 分别为 387 和 307. 比较可知, 本文给出的调度方案较文献 [8] 的结果要优, 这说明本文方法不仅可行有效, 而且具有一定的优越性.

## 5 结　论

本文研究了柔性制造系统的改进粒子群无死锁调度算法. 基于柔性制造系统的 Petri 网模型, 将死锁控制策略嵌入到粒子群算法中, 将粒子位置与系统的基于工件编码的序列相对应, 采用死锁控制策略中的一步向前看方法对调度序列进行安全性验证, 同时将不可行序列转换为可行序列. 在粒子群算法中, 为缓解粒子群算法陷入局部最优解的机率, 并改进算法的优化性能, 本文提出了一种跳出局部极值策略. 仿真结果表明, PSO 对于求解制造系统的无死锁调度问题是有效可行的, 而本文提出的 MPSO 提高了 PSO 的性能, 体现了跳出局部极值的优点.

### 参考文献(References)


[1] Ezpeleta J, Colom J M, Martinez J. A Petri net based deadlock prevention policy for flexible manufacturing systems[J]. IEEE Trans on Robotics and Automation, 1995, 11(2): 173-184.

[2] Reveliotis S A, Lawley M A, Ferreira P M. Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems[J]. IEEE Trans on Automatic Control, 1997, 42(10): 1344-1357.

[3] Xing K Y, Zhou M C, Liu H X, et al. Optimal Petri-net-based polynomial-complexity deadlock-avoidance policies for automated manufacturing systems[J]. IEEE Trans on Systems, Man and Cybernetics, Part A: Systems and Humans, 2009, 39(1): 188-199.

[4] Liu H X, Xing K Y, Zhou M C, et al. Transition cover-based design of petri net controllers for automated manufacturing systems[J]. IEEE Trans on Systems, Man and Cybernetics: Part A, 2014, 43(2): 196-208.

[5] 刘慧霞, 邢科义, 康苗苗. 基于变迁覆盖的制造系统死锁控制策略[J]. 控制理论与应用, 2013, 30(4): 425-431.
(Liu H X, Xing K Y, Kang M M. Transition cover-based deadlock control policies for manufacturing systems[J]. Control Theory & Applications, 2013, 30(4): 425-431.)

[6] Abdallaht I B, Elmaraghy H A, Elmekkawy T. Deadlock-free scheduling in flexible manufacturing system using Petri nets[J]. Int J of Production Research, 2002, 40(12): 2733-2756.

[7] Xu G, Wu Z M. Deadlock-free scheduling strategy for automated production cell[J]. IEEE Trans on Systems, Man, and Cybernetics, Part A, 2004, 34(1): 113-122.

[8] Xing K Y, Han L B, Zhou M C. Deadlock-free genetic scheduling algorithm for automated manufacturing systems based on deadlock control policy[J]. IEEE Trans on Systems, Man, and Cybernetics, Part B, 2012, 42(3): 603-615.

[9] Eberhart R, Kennedy J. A new optimizer using particle swarm theory[C]. Proc of the 6th Int Symposium on Micro Machine and Human Science. Nagoya, 1995: 39-43.

[10] Chen X, Li Y M. A modified PSO structure resulting in high exploration ability with convergence guaranteed[J]. IEEE Trans on Systems, Man and Cybernetics, Part B, 2007, 37(5): 1271-1289.

[11] Lin T L, Horng S J, Kao T W, et al. An efficient job-shop scheduling algorithm based on particle swarm optimization[J]. Expert Systems with Applications, 2010, 37(3): 2629-2636.


(责任编辑: 李君玲)