基于改进的离散PSO算法的FJSP的研究

丁舒阳 黎 冰 侍洪波

(华东理工大学信息科学与工程学院 上海 200237)

摘 要 柔性作业车间调度问题(Flexible Job-shop Scheduling Problem,FJSP)是经典作业车间调度问题的一个扩展,前者更接近于实际生产。以最小化最大完工时间为目标,提出了一种改进的离散粒子群优化算法。传统粒子群优化算法一般适用于优化连续模型问题,FJSP作为复杂度比较高的组合优化问题,是一种典型的离散模型。提出的算法采用机器负荷平衡机制初始化粒子种群,在粒子的更新过程中引入了3个操作算子来更新粒子的工序排序部分和机器分配部分,这3个算子分别为基于工序排序或机器分配的变异、与个体最优位置之间进行工序先后顺序保留的交叉(POX)操作、与全局最优位置进行随机点保存的交叉(RPX)操作。先后执行以上3个算子以完成粒子的一次更新。这种操作能够使种群较快地收敛于最优解。对标准测试案例进行实验的结果表明,所提算法对解决FJSP具有有效性,并且能够快速地搜索到近似最优解;与其他同类算法相比,所提算法在求解效果和收敛速度上均具有优越性。

关键词 作业车间调度,离散优化问题,柔性,粒子群优化

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.04.039

Study on Flexible Job-shop Scheduling Problem Based on Improved Discrete Particle Swarm Optimization Algorithm

DING Shu-yang LI Bing SHI Hong-bo

(School of Information Science and Engineering,East China University of Science and Technology,Shanghai 200237,China)

Abstract Flexible job-shop scheduling problem is an extension of the classical job-shop scheduling problem. The former is much closer to the practical production. Aiming at minimizing the maximum completion time,this paper proposed an improved discrete particle swarm optimization algorithm. The traditional particle swarm optimization algorithm is applicable to optimize the continuous models. As a combinatorial optimization problem with high complexity,FJSP is a typically discrete model. The proposed algorithm utilizes the load balancing mechanism for the machines to initiate the population,and introduces three operators into the procedure of updating the individuals' status in the population. The three operators are respectively described as follows:the mutation based on the operation sequencing or the machine assignment,the precedence preserving order based crossover between current particle and the individual optimal,and randpoint preservation crossover between current particle and the global optimal. A particle is completely updated by using three operators sucessively. This method makes the population converge to the optimal solution very fast. The experimental results of benchmark instances show that the proposed algorithm can practically solve the flexible job-shop scheduling problem and search the near optimal solutions very fast. The proposed algorithm outperforms the other similar algorithms with respect to searching efficiency and convergence speed.

Keywords Job-shop scheduling,Discrete optimization problem,Flexibility,Particle swarm optimization

1 引言

柔性作业车间调度(FJSP)[1]相较于经典车间调度更符合实际生产,因而在近些年引起了广泛的关注。在FJSP中,任意一个工件的任意一道工序都可以在多台机器上加工,选择不同的机器意味着可能产生不同的调度结果,这是因为每道工序在不同机器上的加工时间不尽相同。小规模的FJSP可以通过枚举法寻找最优调度,而大规模的FJSP的复杂度随着机器数与工件数的增加而急剧上升,因此,寻找一种有效的调度方法使问题快速收敛就成为了研究者们需要攻克的难题。

与许多其他的组合优化问题相同,解决FJSP的办法也

到稿日期:2017-01-20 返修日期:2017-03-07

丁舒阳(1989-),女,硕士生,主要研究方向为生产调度领域的智能优化算法,E-mail:jianliyangyang@163.com;黎 冰(1972-),女,博士,副教授,主要研究方向为电气自动控制系统、最优化方法、运动控制系统、生产过程控制等,E-mail:libing@ecust.edu.cn;侍洪波(1965-),男,博士生导师,主要研究方向为工业过程建模、控制与优化、化工过程工况监控、故障检测及故障诊断、生物信息计算与处理、先进控制技术及应用、智能优化方法、过程系统工程,E-mail:hbshi@ecust.edu.cn(通信作者)。

经历了由简单到复杂的过程，例如从开始的数学方法到后来的启发式智能优化算法，前者虽然能保证得到最优解，但求解速度较慢且一般只能解决小规模问题。启发式智能优化算法能弥补数学方法的缺陷，能够解决复杂度高的大规模调度问题且收敛速度快，虽不能保证得到的解是最优的，但达到一定程度的近似最优解通常能够满足实际生产需求。Brandimarte[1]最早将启发式智能优化算法用于求解单目标 FJSP，该算法名为混合禁忌搜索(Tabu Search，TS)算法。近些年，随着对此类问题研究的不断深入，大量的启发式算法被用来解决 FJSP。Kacem 等人[2]提出了一种局部初始化(Approach by Localization，AL)方法，该方法根据机器的负荷来为每道工序分配加工机器，以达到各机器的负载平衡，并且运用了 3 种分配规则对工序进行排序。Pezzella 等人[3]将遗传算法(Genetic Algorithm，GA)用于求解 FJSP，利用许多相关领域的知识来产生初始种群。Xing 等人[4]提出了一种蚁群优化(Ant Colony Optimization，ACO)算法，该算法根据优化性能的好坏动态地调整算法参数，其优化指标是根据最优解更新所需要的迭代次数。Zhang 等人[5]提出了一种改进的遗传算法，该算法用一个两维的向量来表示一个可行解，这个两维向量分别代表机器分配和工序排序；该算法还对文献[2]的 AL 进行了改进，以产生初始种群。为了增强算法的鲁棒性和稳定性，文献[6]提出了一种两阶段混合遗传算法来得到可预测调度，第一阶段以优化最大完工时间为目标，第二阶段包含了其他目标函数并且将机器分配与工序排序融合到解码过程中。文献[7]提出了一种新的染色体编码方法，该方法将染色体分为工序排序和机器选择两部分，并且为交叉和变异设计了不同的策略；实验采用了 8 工件 8 机器的测试案例来验证算法的实用性，结果表明，提出的算法能够有效求解 FJSP。文献[10]提出了一种混合了 TS 的粒子群优化(Particle Swarm Optimization，PSO)算法来求解多目标 FJSP。近些年，随着元启发式(Meta-Heuristic)算法的发展，越来越多的研究者将其应用于 FJSP。Yazdani 等人[8]提出了一种并行变邻域搜索(Variable Neighborhood Search，VNS)算法，该算法基于 AL 的机器分配和随机工序排序的方法产生大量初始解，以最小化最大完工时间(makespan)为目标，利用 9 种邻域结构进行并行搜索。Defersha 等人[9]提出了一种并行遗传算法，以解决复杂的 FJSP，问题的复杂性在于考虑了工序间的转接时间、机器的释放时间以及时滞要求等；该算法基于 Pezzella 等人的 GA，并行性通过小岛模型和随机连接拓扑结构来实现。

针对上述算法中可能产生不可行解以及收敛速度较慢等问题，本文提出了一种改进的离散粒子群优化(Discrete Particle Swarm Optimization，DPSO)算法。不同于一般解决 FJSP 的算法随机地产生初始种群，本文通过运用一种全局搜索(Global Search，GS)、局部搜索(Local Search，LS)和随机搜索(Random Search，RS)相结合的方式，产生一个质量较好的初始种群，从而达到快速收敛的目的。粒子编码方法采用工序排序(Operation Sequencing，OS)和机器分配(Machine Assignment，MA)相结合构成的向量，其在编码形式上不同于传统方法，可以有效地避免非法解的产生。本文改进的 DPSO 在搜索最优解及更新粒子状态的过程中对算法引入的参数做了许多合理的优化，能够保证算法快速收敛。本文在实验仿真部分对标准测试案例进行计算，并与几种相似算法进行比较，实验结果验证了本文算法的可行性。

2 FJSP 问题的描述和评价指标

FJSP 可以描述为 N 个工件在 M 台机器上加工，机器集为 U，每个工件 J_i 包含若干加工顺序不可变的工序 $O_{i,j}\subseteq O_i$（$O_{i,j}$ 表示第 i 个工件的第 j 道工序）。每道工序 $O_{i,j}$ 在其可选加工机器集 $U_{i,j}\subseteq U$ 中选择一台机器进行加工，其中 $U_{i,j}$ 的个数不少于 1。工序 $O_{i,j}$ 在分配到的机器 M_k（$M_k\in U_{i,j}$）上的加工时间为 $p_{i,j,k}$[4]，其中工件 J_i：$\{1\leqslant i\leqslant N\}$，工序 $O_{i,j}$：$\{1\leqslant i\leqslant N, 1\leqslant O_{i,j}\leqslant O_i\}$，对于机器 M_k：$\{1\leqslant k\leqslant M\}$。有关 FJSP 的假设如下：

1)工件 J_i 的各个工序 $O_{i,j}$（$O_{i,j}\subseteq O_i$）必须按照预先定义的顺序进行加工。

2)给工件 J_i（$J_i\in J$，$J=\{J_1,J_2,\cdots,J_N\}$）的工序 $O_{i,j}$ 加工的机器从它的可加工机器集 $U_{i,j}$ 中选择。

3)每台机器在同一时刻只能加工一道工序，并且机器之间的交接时间忽略不计。

4)一个工件的某道工序在同一时刻只能被一台机器加工。

5)同一工件的所有工序按顺序进入车间，同一工件的某道工序只有在其前序工序完成后才能加工。

6)一个工件的某道工序只有当其分配到的机器空闲时才能进行加工。

7)各工序的加工时间和其可选加工机器集是已知的。

对 FJSP 问题求解得到一个调度后，通常需要一个或多个评价指标来判断解的优劣，这样的评价指标通常称为目标函数，又叫适应度值。文献中研究 FJSP 运用得最为广泛的一个评价指标就是最大完工时间(makespan)，即一个可行调度中所有机器最后一道工序加工完成的时间中最大的那个值，它是衡量车间调度性能优劣的根本指标，主要用来体现车间的生产效率，公式描述如式(1)所示：

$$f=\min C_{\max}=\min\{\max_{i=1}^{N}(C_i)\} \tag{1}$$

3 基本 PSO 算法

PSO 在 1995 年由 Kennedy 等人[10]通过研究鸟类的社会行为而提出。研究人员发现，群居的鸟类中的每个成员会根据自身的经验和它与族群中其他成员相互交流获得的信息来决定自身的飞行速度与方向，这个概念是 PSO 算法的主要原理。鸟群中的每个个体又叫作粒子，通过在优化问题的解空间中飞行来搜寻最优解。

假设有一个 D 维的搜索空间和一个包含 N 个粒子的粒子群在其搜索空间内搜寻全局最优解，每个粒子包含 3 个 D 维向量的信息，分别为：位置向量 $X_i=\{X_{i,1},X_{i,2},\cdots,X_{i,D}\}$，速度向量 $V_i=\{V_{i,1},V_{i,2},\cdots,V_{i,D}\}$ 和自身最优位置向量 $P_i=\{P_{i,1},P_{i,2},\cdots,P_{i,D}\}$。在连续的搜索空间中，一个位置向量的每个维度对应于决策变量的一个值，换言之，每个粒子的位置对应于问题的一个潜在解，该粒子的适应度值可以通过把 P_i 中的各个值代入到目标函数中计算得到，适应度值达到的满

意度越大，粒子的位置越好。速度向量代表了粒子在各个维度上经过的距离，自身最优位置向量是各个粒子各自到达过的最佳位置。种群更应当考虑全局最优位置，即在种群的所有粒子中使适应度值最优的那个粒子的最优位置，表示为 $P_g=\{P_{g,1},P_{g,2},\cdots,P_{g,D}\}$。一个粒子的状态由其位置和速度两个因素表征，它们的更新规则分别由式(2)和式(3)表示：

$$V_i^{k+1}=\omega V_i^k+C_1r_1(P_i^k-X_i^k)+C_2r_2(P_g^k-X_i^k) \tag{2}$$

$$X_i^{k+1}=X_i^k+V_i^{k+1} \tag{3}$$

其中，k 表示迭代的次数；i 为种群中的第 i 个粒子；ω 为惯性权重，表示粒子受自身影响的程度，它能够调整粒子的飞行速度，从而使粒子趋于收敛；C_1，C_2 为正的加速常量，分别表示粒子受个体经验和社会经验的影响程度；r_1，r_2 是范围为(0，1)的随机数；P_i^k 表示第 i 个粒子的个体最优位置；P_g^k 表示粒子种群的全局最优位置；X_i^k 表示第 i 个粒子在第 k 代时的位置。

4　改进的 DPSO 算法

由式(2)、式(3)可见，标准的 PSO 算法采用实数编码，使粒子位置在连续空间内变化，进而完成优化操作。有的文献使采用近似取整的方式来编码和解码，使得 PSO 算法适用于 FJSP，但这样会增加算法的求解时间，也有可能引入不可行解。本文采用离散的编码方式对 FJSP 进行编码，使得粒子的位置更新可以直接在离散空间中进行。除此之外，传统随机产生初始种群的方法往往使得初始种群的解的质量偏低，因此本文运用 GS，LS 和 RS 3 种搜索相结合的方式来初始化种群。参数合理与否直接影响着算法的性能，本文采用正交实验设计来优化算法中使用到的参数。

4.1　FJSP 粒子编码方式

本文采用了一种整数编码方式，每个粒子由两部分组成，即工序排序部分(Operations Sequencing，OS)和机器分配部分(Machines Assignment，MS)，它们的长度均为 T_0。OSMA 表示一个粒子的位置向量，其长度为 $2T_0$。表 1 给出了 2 个工件、5 台机器的加工时间表，其中“—”表示该道工序不能在该机器上加工。

表 1　一个 2×5 的 FJSP 实例

Table 1　An example of FJSP with 2×5

工件	工序	可选加工机器及其加工时间				
		M_1	M_2	M_3	M_4	M_5
J_1	O_{11}	2	6	5	3	4
	O_{12}	—	8	—	4	—
J_2	O_{21}	3	—	6	—	5
	O_{22}	4	6	5	—	—
	O_{23}	—	7	11	5	8

注：J_1 表示工件 1；O_{11} 表示工件 1 的第一道工序；表中的数据表示相应工序在该机器上的加工时间

图 1 是对应于表 1 的一个粒子编码，粒子的 OS 和 MA 的长度均为工序总数，OS 的“1”“2”分别代表工件 J_1 和 J_2，第 1 个“1”代表工件 J_1 的第一道工序 O_{11}，第 2 个“1”代表工件 J_1 的第二道工序 O_{12}，其他工件同理。MA 部分按照工序顺序排列，换言之，MA 与 OS 并不是一一对应的，MA 的第 1 个元素表示给工序 O_{11} 分配的机器在其机器集中的顺序数，并非机器数本身，例如此处的“4”就表示 O_{11} 可选择机器集中的第 4 台机器 M_4，第 3 个元素“2”表示给工序 O_{21} 分配其可选加工机器集中的第 2 台机器，查询表 1 可知，这台机器是 M_3 而不是 M_2。采用这样的编码方式是为了避免在后续的粒子更新时产生不可行解。

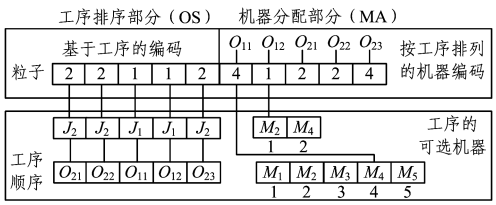

图 1　FJSP 粒子编码

Fig. 1　Particle encoding for FJSP

4.2　改进的 DPSO 算法的各项操作

文献[11]提出对 PSO 算法进行离散化操作，并且将其成功应用于作业车间调度，本文进一步将其应用到 FJSP 中。基于 DPSO，本文重新设计了粒子位置的更新方法，如式(4)所示：

$$X_i^{k+1}=C_2\otimes f_3\{C_1\otimes f_2[\omega\otimes f_1(X_i^k),P_i],P_g\} \tag{4}$$

在式中系数的含义与式(1)一样，f_1，f_2 和 f_3 是 3 个不同的操作算子，$\otimes$ 表示一种运算关系，其针对 FJSP 的具体设计会在本节中详细描述。粒子位置的更新式(4)由 3 部分组成，分别如式(5)－式(7)所示：

$$E_i^k=\omega\otimes f_1(X_i^k)=\begin{cases}f_1(X_i^k), & r<\omega\\ X_i^k, & \text{其他}\end{cases} \tag{5}$$

$$F_i^k=C_1\otimes f_2(E_i^k,P_i)=\begin{cases}f_2(E_i^k,P_i), & r<C_1\\ E_i^k, & \text{其他}\end{cases} \tag{6}$$

$$X_i^{k+1}=C_2\otimes f_3(F_i^k,P_g)=\begin{cases}f_3(F_i^k,P_g), & r<C_2\\ F_i^k, & \text{其他}\end{cases} \tag{7}$$

在以上 3 个公式中，r 都表示(0，1)之间的随机数。式(5)表示粒子对先前状态的思考，f_1 相当于对粒子自身进行变异操作，惯性系数 ω 相当于变异概率。式(6)表示经过式(5)的操作后，粒子通过自身最优位置进行调整，f_2 表示 E_i^k 与 P_i 进行交叉操作，C_1 为其交叉概率。式(7)表示经过式(6)的操作后，粒子通过全局最优位置进行调整，f_3 表示 F_i^k 与 P_g 进行交叉操作，C_2 为其交叉概率。

4.2.1　$f_1(X_i^k)$操作算子

由于 DPSO 实际上也是进化算法中的一种，与其他进化算法一样也存在早熟收敛问题，即使得种群收敛于局部极值从而搜索停滞。为了避免算法陷入局部吸引，适当增加种群的多样性，扩大搜索范围是一种常用手段。Coallo 等人[12]将变异机制引入 PSO 算法中。受此方法的启发，本节的 f_1 算子采用基于粒子编码的两种变异操作，分别对粒子向量的 OS 和 MA 进行调整，其实现方法分别如下：

1) 基于 OS 的变异。以图 1 为例，由于采用这种编码方式 OS 中各元素以任意的顺序排列都不会产生非法解，因此使用该变异操作对 OS 进行随机排序。例如，图 1 中 OS 原本的顺序是{2，2，1，1，2}，经过变异后顺序变为{2，1，2，2，1}，而 MA 原本就是按照 $O_{11}\rightarrow O_{12}\rightarrow O_{21}\rightarrow O_{22}\rightarrow O_{23}$ 的顺序排列的，因此 MA 不需要调整。

2) 基于 MA 的变异。由于每道工序都可以被不少于一

台机器加工，因此每道工序都有一个可加工机器集。基于MA的变异操作步骤如下：随机选择OS上的某一维数据，如第二维，其值为“2”，这是OS中的第二个“2”，代表工序 O_{22}，其对应的加工机器在MA上对应第四维数据“2”，也即 O_{22} 分配了其加工机器集 U_{22} 中的第二台机器 M_2，然后从 U_{22} 中随机选择另一台机器来代替MA中的机器，特别需要注意的是，MA中的元素是机器的顺序号而非机器号。

采用轮盘赌的方式选择以上两种变异中的一种作为 $f_1(X_i^k)$ 操作算子。

4.2.2 $f_2(E_i^k, P_i)$ 操作算子

$f_2(E_i^k, P_i)$ 表示当前粒子向个体最优位置 P_i 学习的过程，本节采用改进后的保留工序先后顺序的交叉（Precedence Preserving Order based Crossover，POX）[11] 作为 f_2 操作算子。本文的粒子是一个由OS和MA组成的一维向量，通过将其拆分成OS与MA两个部分后做简单的改进，就可以使POX适用于FJSP。改进后的POX的操作步骤为：将工件集 $\{1,2,3,\cdots,N\}$ 随机地分为两个非空互补子集 $\mathcal{J}_1$ 和 $\mathcal{J}_2$，针对 E_i^k 和 P_i 的OS部分，依次复制 E_i^k 中属于 $\mathcal{J}_1$ 的工件号到子代 F_i^k 中，同时复制 P_i 中属于 $\mathcal{J}_2$ 的部分到子代 F_i^k 中，复制到 F_i^k 中的顺序是按照各工序在各自原本的OS中的顺序进行排序的，若遇到顺序号一样的工序，则随机将两者按相邻的工序进行排列。对于MA而言，则只需要把 E_i^k 和 P_i 中各自选中的工序对应的机器顺序号复制到 F_i^k 的MA中即可。

图2表示一个3×4的FJSP种群中一个粒子的POX交叉过程，即把当前粒子 E_i^k 中的工件2复制到子代 F_i^k 中，将个体最优位置 P_i 中的工件1和工件3复制到子代 F_i^k 中，工序顺序与其在各自原先粒子中的顺序相同。值得指出的是，为了显示得更直观，将OS与MA一一对应起来，而实际编码中MA按4.1节所示的方式编码。

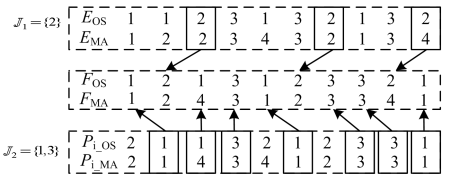

图2 POX的交叉过程

Fig. 2 Crossover procedure of POX

4.2.3 $f_3(F_i^k, P_g)$ 操作算子

结合FJSP的特征，本文采用了一种新的随机点保留交叉（Rand-point Preservation Crossover，RPX）算子来调整粒子的机器选择部分。RPX操作算子的执行过程如下：将 F_i^k 中的OS部分复制到 X_i^{k+1} 中，随机生成一个各元素范围为(0,1)的一维向量 R，其长度为粒子中OS部分的长度 T_0。将 R 的各元素与OS的各元素对应起来，找出向量 R 中元素小于 pf 的位置，记录 F_i^k 中对应位置的工序，然后将 P_g 中相同位置的工序的机器顺序号复制到 X_i^{k+1} 中对应的MA中，其他工序号对应的机器顺序号不变。

对4.2.2节的同一个粒子进行RPX操作的过程如图3所示，假设当前 $pf=0.8$，随机数组 R 中小于0.8的元素均被选中，例如第7个元素 $R_7=0.5<pf$，这一位置对应当前粒子 F_i^k 中的工序 O_{22}，那么找出 P_g 中的工序 O_{22}，对应于 P_{g_OS} 第5位，该工序在 P_g 中对应的机器顺序编号为4，那么在子代 FF_i^k 中便将第4台机器分配给工序 O_{22}，取代原本的第2台机器。机器显示与4.2.2节同理。

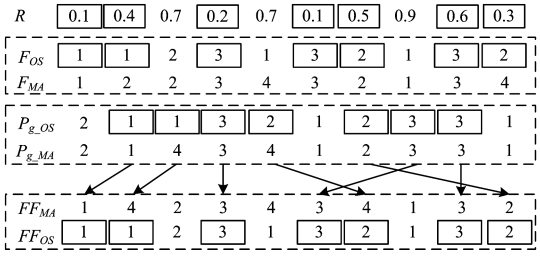

图3 RPX交叉过程

Fig. 3 Crossover procedure of RPX

pf 为自适应调整概率，其计算如式(8)所示：

$$pf = pf_{\max} - \frac{pf_{\max} - pf_{\min}}{Iter} \times k \tag{8}$$

其中，$Iter$ 为算法的总迭代次数，k 为当前迭代次数，$pf_{\max}$ 和 $pf_{\min}$ 分别为预先给定的最大调整概率和最小调整概率。由式(8)可以看出，在算法搜索的初期，由于 k 值比较小，因此 pf 的值比较大，粒子个体中就有较多的元素与 P_g 进行交叉操作，从而有利于快速地向全局最优靠拢。随着搜索代数的增加，k 值增大，pf 值减小，粒子中被选中与 P_g 进行交叉的元素减少，这样有利于局部的精细搜索。

4.3 改进的DPSO算法的流程

将改进的DPSO算法用于求解FJSP，具体步骤如下：

1）确定最优参数：包括种群规模 pop 以及算法中用到的常数（惯性常量 ω、学习因子 C_1 和 C_2、调整概率 $pf_{\max}$ 和 $pf_{\min}$、总的迭代次数 $Iter$）。

2）种群初始化：设置当前运行代数 $k=0$，按照GS、LS和RS相结合的方法得到数量为 pop 的初始粒子群，按照3种方法得到的粒子数量进行平均分配。然后计算每个粒子的目标函数值，初始化个体最优位置 P_i 和全局最优位置 P_g。

3）搜索操作：根据式(5)完成粒子的初步更新，采用轮盘赌的方法决定选择4.2.1节中的两种算子。

4）根据式(6)，采用4.2.2节中的操作算子完成粒子的二次更新。

5）根据式(7)，采用4.2.3节中的操作算子完成粒子位置的一次完整更新。

6）对种群中的每个粒子执行步骤3)－步骤5)的更新操作，计算当前种群中各粒子的适应度值，并通过比较更新个体位置最优 P_i 和全局最优位置 P_g。

7）进行迭代：$k=k+1$，判断是否满足终止条件 $k \geqslant Iter$，若满足终止条件则停止迭代，输出全局最优位置 P_g，并以此作为搜索到的最优调度，否则转步骤3)。

算法1 改进的DPSO算法

1. Clear all;

2. Input arguments: pop=100, ω=0.15, C_1=0.5, Iter=100, C_2=0.7, pf_{min}=0.2, pf_{max}=0.8;

% Initialize the population by GS, LS and RS respectively%

3. for i=1:pop/3

```
4.    x=global_selection();// x is a particle in the population
5.  end
6.  for i=pop/3+1:pop/3 * 2
7.    x=local_selection();
8.  end
9.  for i=pop/3 * 2+1:pop
10.   x=random_selection();
11. end
    % Get initialized local and global optimum solution%
12. pB_matrix=initial_population;
13. calculate the makespan value for each pB;
14. if makspan of pB* is minimum
15.   gB=pB* ;
16. end
    %The outer for loop to iterate%
17. for generation=1:Iter
    %The inner for loop to traverse the population%
18.   for i=1:pop
    %First perform the f1 operator%
19.     if r1<ω
20.       [E(i,:)]=mutation_OS(initial_population(i,:));
21.       or [E(i,:)]=mutation_MA();
22.       else E(i,:) =initial_population(i,:);
23.     end
        %Second perform the f2 operator%
24.     if r2<C1
25.       [F(i,:)]=POX_crossover(E(i,:),pB_arr(i,));
26.     else F(i,:)=E(i,:);
27.     end
        %Finally perform the f3 operator%
28.     if r3<C2
29.     [initial_population(i,:)]=RPX_crossover(F(i,:),gB);
30.     else initial_population(i,:)=F(i,:);
31.     end
        % Renew the local optimum solutions%
32.     makespan_new=decoding(initial_population(i,:));
33.     makespan_local=decoding(pB_matrix(i,:));
34.     if makespan_new < makespan_local
35.       pB_matrix(i,:)=initial_population(i,:);
36.     end
37.   end
      % Renew the global optimum solution
38.   if makspan of pB* is minimum
39.     gB=pB* ;
        %pB* belongs to pB_matrix %
40.   end
41. end
42. return gB and the makespan value of gB;
```

4.4　算法复杂度分析

本文算法的时间复杂度主要与种群规模 pop、粒子长度 $2T_0$ 以及总的运行迭代次数 $Iter$ 密切相关，在 4.3 节针对粒子更新过程的步骤 3)－步骤 5)中，步骤 3)对 MA 和 OS 分别进行 1 次调整，并且这两种选择方式是互斥的，步骤 4)对 OS 和 MA 各进行 1 次调整，步骤 5)对 MA 进行 1 次调整。粒子更新对 OSMA 进行调整的方式有两种，一种是对 MA 进行 3 次调整并对 OS 进行 1 次调整，另一种是对 MA 和 OS 分别进行 2 次调整，不管是哪一种调整方式，其时间复杂度都为 $O(pop\times 4T_0)$。步骤 6)中计算粒子的目标函数值的时间复杂度为 $O(pop\times 2T_0)$，更新全局最优解的时间复杂度为 $O(pop)$。再考虑算法的总迭代次数，该算法总时间的复杂度等于各粒子自更新的复杂度、计算目标函数值的复杂度及更新全局最优解时各粒子与全局最优解进行比较的时间复杂度这三者之和，再乘以迭代总数，其计算如式(9)所示：

$$Iter\times[O(pop\times 4T_0)+O(pop\times 2T_0)+O(pop)] \quad (9)$$

5　仿真实验与算法评估

本文仿真实验的硬件环境为 Intel(R) Core(TM) i7-6500U CPU @ 2.50 GHz，RAM 8 GB；软件环境为 Windows 10(基本版)操作系统。采用仿真平台 Matlab 2016a 编写算法。

5.1　算法参数的选择

参数的大小对算法的性能有着较大的影响，因此需要合理配置各类参数。对 DPSO 算法性能影响较大的主要参数有惯性常量 ω、学习因子 C_1 和 C_2、调整概率 $pf_{\max}$ 和 $pf_{\min}$ 共 5 个。本文采用正交实验设计[13]来配置最佳参数组合，文中 5 个参数分别被设置为 5 个因素，每个因素设置 4 个水平，根据正交实验设计的原理，只需要测试 $16(4^2=16)$ 组参数组合就能找到最佳的参数配置。本文采用 FJSP 标准测试案例 MK03[1]进行正交实验以调节参数，正交实验表 $L_{16}(4^5)$ 及其实验结果如表 2 所列。

表 2　$L_{16}(4^5)$正交表及实验结果

Table 2　Orthogonal table by $L_{16}(4^5)$ and experimental results

实验号	因子					实验结果(makespan)
	ω	C_1	C_2	$pf_{\max}$	$pf_{\min}$	
	1	2	3	4	5	
1	0.10(1)	0.30(1)	0.60(1)	0.80(1)	0.10(1)	254
2	0.10(1)	0.40(2)	0.70(2)	0.85(2)	0.15(2)	251
3	0.10(1)	0.50(3)	0.80(3)	0.90(3)	0.20(3)	265
4	0.10(1)	0.60(4)	0.90(4)	0.95(4)	0.25(4)	285
5	0.15(2)	0.30(1)	0.70(2)	0.90(3)	0.25(4)	242
6	0.15(2)	0.40(2)	0.60(1)	0.95(4)	0.20(3)	224
7	0.15(2)	0.50(3)	0.90(4)	0.80(1)	0.15(2)	219
8	0.15(2)	0.60(4)	0.80(3)	0.85(2)	0.10(1)	261
9	0.20(3)	0.30(1)	0.80(3)	0.95(4)	0.15(2)	274
10	0.20(3)	0.40(2)	0.60(1)	0.90(3)	0.10(1)	248
11	0.20(3)	0.50(3)	0.90(4)	0.85(2)	0.25(4)	245
12	0.20(3)	0.60(4)	0.70(2)	0.80(1)	0.20(3)	244
13	0.25(4)	0.30(1)	0.90(4)	0.85(2)	0.20(3)	240
14	0.25(4)	0.40(2)	0.80(3)	0.80(1)	0.25(4)	260
15	0.25(4)	0.50(3)	0.70(2)	0.95(4)	0.10(1)	237
16	0.25(4)	0.60(4)	0.60(1)	0.90(3)	0.15(2)	260
k_1	263.75	252.50	246.50	**240.50**	250.00	
k_2	**232.75**	245.75	**243.50**	249.50	247.25	
k_3	253.00	**238.00**	265.00	253.75	**243.25**	
k_4	249.25	262.50	243.75	255.00	258.25	
SD	**12.8556**	10.3951	10.2984	6.5618	6.3455	

表 2 列出了 5 个因子各自在 4 个不同水平下的实验结

果，这里用目标函数 *makespan* 来评估，*makespan* 取每组参数组合运行 5 次的平均值。在表 $2k_1-k_4$ 这 4 行数据中，第 j 列的 k_i 值表示参数 j 在水平 i 上的 4 组实验结果的平均值，例如 ω 例的 k_1 值表示参数 ω 在水平 1($\omega=0.1$)时 4 组 *makespan* 的平均值，找出每一列中对应最小的 k_i 值，表明该因子所对应的 i 水平参数值最好。表中 k_2 的值最小，表明 w 应选择水平为 2 时所对应的值，即 $w=0.15$。第 j 列的 *SD* 值表示参数 j 从 k_1-k_4 的标准差(Standard Deviation，SD)。横向比较各 *SD* 值，值越大表明该因子的变化对算法性能的影响越大。表 2 的实验结果表明，ω 参数的变化对算法性能的影响最大。

根据实验测试结果，如表 2 中黑体加粗部分所示，本文算法选择以下参数：$\omega=0.15$，$C_1=0.5$，$C_2=0.7$，$pf_{\max}=0.8$，$pf_{\min}=0.2$。其他参数选择如下：迭代次数 $Iter=100$，种群规模 $pop=100$，每个测试案例独立运行 10 次，实验结果取平均值。

5.2 仿真与分析

最大完工时间 *makespan* 可以通过甘特图[14]直观地显示，图 4 是应用本文算法得到的实例 MK01 的最优解甘特图(基于软件 Matlab 运行结果绘制)。图中横坐标为加工时间，纵坐标为机器号，同一色块代表同一工件的不同工序，在机器 1 上的第一个色块的标识 $p(601)=3$ 表示工件 6 的第 1 道工序在机器 1 上加工，其加工时间为 3，其余类似。甘特图不仅能够反映各工序在机器上的分配情况，还能反映各工序之间的约束关系。从图 4 可以看出，在当前调度下，实例 MK01 的最大完工时间 *makespan* 为 42。

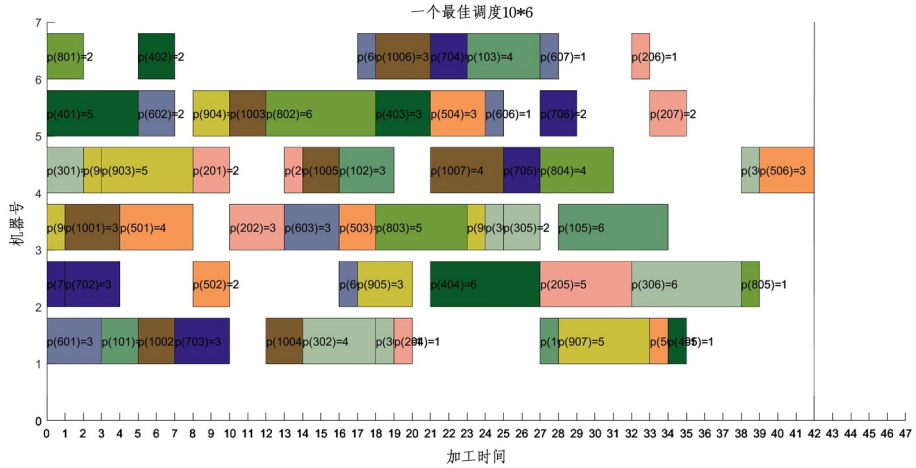

图 4 实例 MK01 的最优调度甘特图

Fig. 4 Gantt chart of optimal schedule for MK01 instance

为了验证算法在求解 FJSP 问题方面的有效性与优越性，选取了研究较为普遍的 Brandimarte[1]标准测试案例中的前 10 个案例和 Kacem[2]中的 5 个案例进行实验。其中文献[1]给出的标准测试案例共有 15 个，工件数的取值范围为 10～30，机器数的取值范围为 4～15，所有 Brandimarte 实例均属于部分 FJSP 的范畴。由于现有文献鲜有对后 5 个案例进行测试，因此，本文不列出其数据，但可以证明对于这 5 个规模较大的案例，本文算法全方面优于文献[1]中的算法。文献[2]给出的测试案例的规模都比较小，并且所有 Kacem 实例均属于完全 FJSP 的范畴。通过最大完工时间 *makespan* 这个性能指标将本文算法与文献[15]中的启发式算法(Heuristic Algorithm，HA)以及文献[16]中的并行混合元启发式(Parallel Hybrid Meta Heuristics，PHMH)算法的实验结果进行比较，如表 3 所列。

表 3 基于 Brandimarte 和 Kacem 标准测试案例的各算法的比较

Table 3 Comparison between different algorithms based on Brandimarte and Kacem benchmark instances

实例	$n\times m$	HA[15]				PHMH[16]			本文的 DPSO		
		C^*	$C_{\max}$	*RPD*	*Time*	$C_{\max}$	*RPD*	*Time*	$C_{\max}$	*RPD*	*Time*
MK01	10×6	36	42	16.67	90.00	41	13.89	47.87	42	16.67	50.12
MK02	10×6	24	28	16.67	17.00	30	25.00	36.12	32	33.33	33.40
MK03	15×8	204	204	0.00	52.00	204	0.00	330.10	204	0.00	300.89
MK04	15×8	48	75	56.25	20.00	65	35.42	115.22	80	66.67	76.56
MK05	15×4	168	179	6.55	20.00	174	3.57	106.12	173	2.98	93.30
MK06	10×15	33	69	109.09	450.00	71	115.15	2119.53	66	100.00	1960.80
MK07	20×5	133	149	12.03	39.00	148	11.28	112.20	145	9.02	106.83
MK08	20×10	523	555	6.12	66.00	551	5.35	998.05	524	0.19	889.16
MK09	20×10	299	342	14.38	940.00	410	37.12	1286.91	364	21.74	981.73
MK10	20×15	165	242	46.67	120.00	267	61.82	649.15	252	52.73	590.82
Kac1	4×5	11	11	0.00	1.50	—	—	—	11	0.00	3.20
Kac2	8×8	14	15	7.14	9.40	—	—	—	15	7.14	13.50
Kac3	10×7	11	13	18.18	14.10	—	—	—	12	9.09	20.65
Kac4	10×10	7	7	0.00	2.18	—	—	—	7	0.00	5.88
Kac5	15×10	11	12	9.09	5.32	—	—	—	11	0.00	7.90

注：Kac 是 Kacem 案例的缩写形式

表 3 中的 n 为工件数，m 为机器数，“－”表示文献中未给出该参数的有关信息。C^* 表示文献中该案例目前搜寻到的最好的最大完工时间的平均值。由于每个实例并不完全一样，而且各类算法在不同的软硬件平台上实现，因此这里选用了最优解平均值。$C_{\max}$ 表示算法各自搜寻到的最优解；$Time$ 表示各算法在其各自软硬件平台上的运行时间，单位为秒；RPD 为各算法相对于 C^* 的百分比偏差值（Relative Percent Deviation，RPD），表示各算法搜寻到的最优值与 C^* 的相近程度，其计算公式如式(10)所示：

$$RPD=\frac{C_{\max}-C^*}{C^*}\times 100 \tag{10}$$

由表 3 可以看出，相对于文献[15]的 HA，就 $C_{\max}$ 而言，针对规模较小的 MK01－MK04，改进的 DPSO 略逊色于 HA，但是对于规模较大且复杂度较高的 MK05－MK10，DPSO 则全面优于 HA，这说明改进的 DPSO 适合求解规模较大且复杂度较高的 FJSP，这是因为改进的 DPSO 采用的针对 OS 和 MA 的变异机制使得种群更容易跳出局部极优解，从而增强了种群向着全局最优解方向搜寻的能力，尤其是对于案例 MK08，DPSO 搜寻到的最优解目标值十分接近 C^*。针对小规模的 Kac1－Kac5，本文搜寻到的最优解全面优于 HA，这说明改进的 DPSO 在可选机器集较多时求解十分有效。在运行时间方面，改进的 DPSO 却不如 HA，这是因为除去软硬件平台的差异后 HA 给出的进化操作十分简便，即使初始种群比较分散，其也能迅速收敛于极优解；而 DPSO 的操作算子比较复杂，不仅对 f_1 算子存在轮盘赌选择，而且针对 f_2 和 f_3 还存在对粒子的某些基因位进行随机选择的操作，因此 DPSO 的收敛速度慢于 HA。由此可见，相较于 HA，DPSO 以较高的时间复杂度为代价，取得了较好的全局寻优能力。

由于文献[16]没有给出 Kacem 测试案例的相关数据，因此不予讨论。对于 Brandimarte 的 10 个案例，就 $C_{\max}$ 而言，除了 MK02 和 MK04 以外，改进的 DPSO 全面优于 PHMH；在运行时间方面，改进算法的收敛速度略快于 PHMH，因为其除去了不同软硬件平台对计算时间造成的差异，尽管改进算法对粒子的操作过程比较复杂，但由于粒子群算法本身具有快速收敛的特性，因此改进的 DPSO 仍然较 PHMH 的收敛速度更快。由此可见，除了个别算例外，改进的 DPSO 在寻优能力和运行时间方面均优于 PHMH。

图 5、图 6 分别给出了在实例 MK06 和 MK08 上改进算法与文献[17]提出的标准遗传算法（Standard Genetic Algorithm，SGA）和改进的遗传算法（Improved Genetic Algorithm，IGA）的收敛曲线对比，结果均反映出改进的 DPSO 的初始种群质量优于 SGA 和 IGA，并且在前 10 次迭代的过程中快速收敛，在 50 次迭代时收敛于最优解；而 SGA 和 IGA 虽然在刚开始时也迅速收敛，但由于初始种群质量较差，随着迭代次数的增加，收敛速度变慢，迭代到 80 次时才逐渐趋于收敛，这说明本文将 GS，LS 和 RS 相结合的种群初始化方法在算法执行的第一步就使种群中出现了较多接近最优解的粒子，即初始种群质量较好。图 5、图 6 表明，改进的 DPSO 的收敛速度较 SGA 和 IGA 更快，收敛效果更佳。

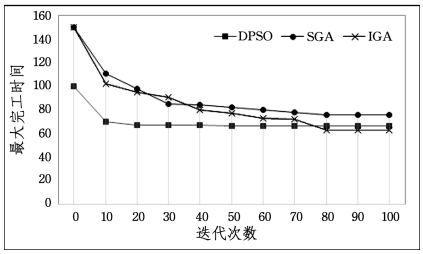

图 5　DPSO 与 SGA 和 IGA 的收敛曲线比较（MK06）

Fig. 5　Comparison of convergence curves between DPSO, SGA and IGA(MK06)

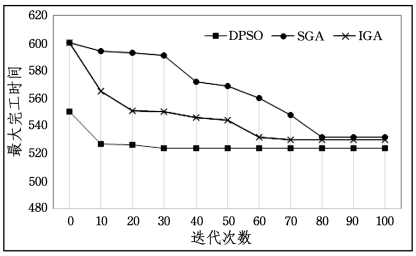

图 6　DPSO 与 SGA 和 IGA 的收敛曲线比较（MK08）

Fig. 6　Comparison of convergence curves between DPSO, SGA and IGA(MK08)

结束语　本文提出了一种解决 FJSP 的改进的 DPSO 算法，通过采用尽量使机器负荷平衡的机制初始化种群，弥补了随机初始化种群的缺陷，这种机制使得初始种群质量较好。利用粒子群算法收敛速度快的特点，将 3 种改进的操作算子引入该算法使其离散化，不仅可以保持种群的多样性，还可以使种群中各个粒子向着较优的粒子靠近。除此之外，本文还使用正交实验设计的方法优化了算法中存在的参数，使得算法性能最佳。与文中所列的其他解决该问题的优化算法相比，本文提出的算法处理多数测试算例时能搜索到更优的解，并且收敛速度更快。根据测试结果可知，算法并非对所有的算例都达到了 C^* 值，这说明跳出局部极优的力度不够大，这是算法存在的不足之处。因此，如何使种群快速地跳出局部极优解并向着全局最优收敛是今后的研究方向。

参 考 文 献

[1] BRANDIMARTE P. Routing and Scheduling in a Flexible Job Shop by Tabu Search [J]. Annals of Operations Research, 1993, 41(3): 157-183.

[2] KACEM I, HAMMADI S, BORNE P. Approach by Localization and Multi-Objective Evolutionary Optimization for Flexible Job-Shop Scheduling Problems [J]. IEEE Transactions on Systems, Man and Cybernetics, Part C(Applications and Reviews), 2002, 32(1): 1-3.

[3] PEZZELLA F, MORGANTI G, CIASCHETTI G. A Genetic Algorithm for the Flexible Job-Shop Scheduling Problem [J]. Computers & Operations Research, 2008, 35(10): 3202-3212.

[4] XING L N, CHEN Y W, WANG P, et al. A Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems [J]. Applied Soft Computing, 2010, 10(3): 888-896.

（下转第 256 页）

ty,2009:1024-1035.

[5] FANG W W,XIE W,HUANG H B,et al. Sequential pattern mining based on privacy preserving [J]. Computer Science, 2016,43(12):195-199. (in Chinese)
方炜炜,谢伟,黄宏博,等. 基于隐私保护的序列模式挖掘[J]. 计算机科学,2016,43(12):195-199.

[6] DONG G,LI J. Efficient mining of emerging patterns:Discovering trends and differences[C]// Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM,1999:43-52.

[7] GHOSH S,FENG M,NGUYEN H,et al. Risk prediction for acute hypotensive patients by using gap constrained sequential contrast patterns [C]// AMIA Annual Symposium Proceedings American Medical Informatics Association. 2014:1748-1754.

[8] JI X,JAMES B,DONG G. Mining minimal distinguishing subsequence patterns with gap constraints [J]. Knowledge Information Systems,2007,11(3):259-286.

[9] WANG X,DUAN L,DONG G,et al. Efficient mining of density-aware distinguishing sequential patterns with gap constraints [C]// 19th International Conference of Database Systems for Advanced Applications. 2014:372-387.

[10] YANG H,DUAN L,HU B,et al. Mining top-k distinguishing sequential patterns with gap constraint [J]. Journal of Software,2015,26(11):2994-3009. (in Chinese)
杨皓,段磊,胡斌,等. 带间隔约束的 Top-k 对序列模式挖掘[J]. 软件学报,2015,26(11):2994-3009.

[11] WANG H F,DUAN L,ZUO J,et al. Efficient mining of distinguishing sequential patterns without a predefined gap constraint [J]. Journal of Computer, 2016, 39(10): 1979-1991. (in Chinese)
王慧锋,段磊,左劼,等. 免预设间隔约束的对比序列模式高效挖掘[J]. 计算机学报,2016,39(10):1979-1991.

[12] WU Y X,WU X D,JIANG H,et al. A heuristic algorithm for MPMGOOC [J]. Journal of Computers, 2011, 34(8): 1452-1462. (in Chinese)
武优西,吴信东,江贺,等. 一种求解 MPMGOOC 问题的启发式算法[J]. 计算机学报,2011,34(8):1452-1462.

[13] WU Y,TANG Z,JIANG H,et al. Approximate Pattern Matching with Gap Constraints[J]. Journal of Information Science, 2016,42(5):639-658.

[14] WU Y,FU S,JIANG H,et al. Strict approximate pattern matching with general gaps[J]. Applied Intelligence,2015,42(3): 566-580.

[15] WU Y,WANG L,REN J,et al. Mining sequential patterns with periodic wildcard gaps[J]. Applied Intelligence,2014,41(1):99-116.

(上接第 239 页)

[5] ZHANG G,GAO L,SHI Y,et al. An Effective Genetic Algorithm for the Flexible Job-Shop Scheduling Problem [J]. Expert Systems with Application,2011,38(4):3563-3573.

[6] AL-HINAI N,ELMEKKAWY T Y. Robust and Stable Flexible Job Shop Scheduling with Random Machine Breakdowns Using a Hybrid Genetic Algorithm [J]. International Journal of Production Economics,2011,132(2):279-291.

[7] WANG J,CHU K. An Application of Genetic Algorithms for the Flexible Job-Shop Scheduling Problem [J]. International Journal of Advancements in Computing Technology, 2012, 4(3):271-278.

[8] YAZDANI M,AMIRI M,ZANDIEH M. Flexible Job-Shop Scheduling with Parallel Variable Neighborhood Search Algorithm [J]. Expert Systems with Application,2010,37(1):678-687.

[9] DEFERSHA F M,MINGYUAN C. A Parallel Genetic Algorithm for a Flexible Job-Shop Scheduling Problem with Sequence Dependent Setups [J]. International Journal of Advanced Manufacturing Technology,2010,49(1/4):263-279.

[10] KENNEDY J,EBERHART R C. A Discrete Binary Version of the Particle Swarm Algorithm [C]// IEEE International Conference on Systems. 2002:4104-4108.

[11] CEHN J,PAN Q K. A Discrete Particle Swarm Optimization Algorithm for Independent Task Scheduling Problem [J]. Computer Engineering,2008,34(6):214-215,218. (in Chinese)
陈晶,潘全科. 求解独立任务调度的离散粒子群优化算法[J]. 计算机工程,2008,34(6):214-215,218.

[12] COELLO C A C,PULIDO G T,LECHUGA M S. Handling multiple objectives with particle swarm optimization[J]. IEEE Transactions on Evolutionary Computation, 2004, 8(3): 256-279.

[13] MONTGOMERY D C. Design and Analysis of Experiments [M]. New York:John Wiley & Sons,2008.

[14] TANG X X,HE W P,HE Y L,et al. Research and Implementation of Job Shop Scheduling System [J]. Aeronautical Manufacturing Technology,2011(5):69-73. (in Chinese)
唐欣欣,何卫平,和延立,等. 面向作业车间的调度系统研究与实现[J]. 航空制造技术,2011(5):69-73.

[15] ZIAEE M. A Heuristic Algorithm for Solving Flexible Job Shop Scheduling Problem [J]. The International Journal of Advanced Manufacturing Technology,2014,71(1-4):519-528.

[16] WOJCIECH B,MARIUSZ U,MIECZYSLAW W. Parallel Hybrid Metaheuristics for the Flexible Job Shop Problem [J]. Computers & Industrial Engineering,2010,59(2):323-333.

[17] ZHANG T N,HAN B,YU B,et al. Flexible Job Shop Scheduling Optimization with Production Capacity Constraints [J]. System Engineering-Theory & Practice,2011,31(3):505-511. (in Chinese)
张铁男,韩兵,于渤,等. 生产能力约束条件下的柔性作业车间调度优化[J]. 系统工程理论与实践,2011,31(3):505-511.