文章编号: 1001-0920(2011)04-0535-05

大规模无等待流水调度问题的邻域迭代搜索算法

宋存利[1a,2], 刘晓冰[1b], 王 伟[1a]

(1. 大连理工大学 a. 电子与信息工程学院，b. 管理学院，辽宁 大连 116024；2. 大连交通大学 软件学院，辽宁 大连 116052)

摘 要: 在分析大规模无等待流水调度问题特点的基础上, 提出了利用相邻工件间完工时间距离求最小化完工时间的方法; 通过研究工件插入和工件对的交换对最小化完工时间的影响, 提出一种邻域迭代搜索算法, 该算法降低了求解完工时间的时间复杂度, 大大提高了算法效率; 为避免算法在邻域搜索过程中陷入局部最优, 将变邻域结构算法的思想应用于其中. 仿真结果表明, 所提出的算法能高效率解决大规模无等待流水调度问题, 所得结果令人满意.

关键词: 邻域迭代搜索算法；无等待流水调度；最小化完工时间

中图分类号: TP301 **文献标识码:** A

An iterative neighborhood search algorithm for large scale no-wait flow-shop

SONG Cun-li[1a,2], *LIU Xiao-bing*[1b], *WANG Wei*[1a]

(1a. School of Electronic and Information Engineering，1b. College of Management，Dalian University of Technology, Dalian 116024，China；2. Software Institute，Dalian Jiaotong University，Dalian 116052，China. Correspondent: SONG Cun-li，E-mail：scunli@163.com)

Abstract: According the characteristic of large-scale no-wait flow shop, a method for calculating makespan is proposed for large-scale no-wait flow shop scheduling by using the distance of adjacent workpiece. At the same time, an iterative neighborhood search algorithm is proposed, which reduces the time complexity, so the efficiency is greatly improved. In order to avoid falling into local optimum, the mind of variable neighborhood search algorithm is used, so that the probability to finding the global optimal solution is enhanced. Experiment results show that the algorithm can solve the problem of large-scale no-wait flow-shop scheduling efficiently, and the result is better.

Key words: iterative neighborhood search algorithm；no-wait flow-shop scheduling；makespan

1 引 言

自 20 世纪 50 年代以来, 生产调度问题一直是学术界研究的热点之一, 其任务是为一系列生产任务在不同的加工设备上安排加工顺序, 以达到某个性能最优, 例如: 最小化最大完工时间、设备利用率达到最高、生产制造成本最低等. 无等待流水调度问题是其中一类典型的约束流水调度问题, 广泛地应用于炼钢、食品加工、化工和制药等工业领域.

最小化完工时间的无等待流水调度问题记为 Fm|nwt|Cmax, 是一类典型的 NP 难题, 很多学者对此进行了研究. 例如: Van Deman 等人[1]提出了分支定界法来寻求最优方案, 并提出了一系列过程以产生最优值. Gdamgadjaran 等人[2]和 Rajendran[3]分别提出了启发式算法 GR 和 RAJ, 实验表明, GR 和 RAJ 均优于由 Bonney 等人[4]和 King 等人[5]提出的启发式算法. 潘全科等人[6]提出了一种将离散化的粒子群算法 (DPSO) 和邻域搜索算法相结合的算法, 实验表明, 该算法的运行效率和所达到最优解的性能都较好. 这些算法对中小规模的流水调度问题非常有效, 然而当问题规模扩大时往往以时间为代价, 即寻优时间太长, 很难被实际应用接受.

本文在研究无等待流水车间调度问题一般规律的基础上, 提出了基于邻域迭代的搜索算法. 该算法利用相邻工件间的完工时间距离求解 Makespan, 将计算的时间复杂度降低一阶; 同时邻域解的求法采用增量计算, 又将计算的时间复杂度降了一阶. 实验表明,

收稿日期: 2010-01-18；**修回日期:** 2010-05-03.

基金项目: 国家自然科学基金项目(70572098).

作者简介: 宋存利(1975−), 女, 博士生, 从事生产作业调度、智能算法等研究；刘晓冰(1956−), 男, 教授, 博士生导师, 从事先进制造模式等研究.

该算法对大规模无等待流水调度问题具有较高的求解效率, 比较适应生产实际; 同时将变邻域搜索思想应用于算法设计中, 避免了算法陷入局部最优. 仿真结果表明, 该算法不仅效率高, 而且在求解质量上与DPSO和TS+M[7]算法接近.

2 问题描述

无等待流水车间调度问题(NWFS)可描述为: n个工件$\{J_1, J_2, \cdots, J_n\}$在给定$m$台设备$\{M_1, M_2, \cdots, M_m\}$上以相同的顺序加工, 同时约定一个工件在某一时刻只能被一台设备加工, 且一台设备在某一时刻只能加工一个工件; 一个工件一旦开始加工便不允许等待, 即必须连续加工完该工件的所有工序才能停止; 各工件在设备上的加工时间已知. 问题是: 如何安排生产, 在满足约束的前提下, 使得n个工件的完工时间达到最小.

n个工件在m台设备上加工, 可能的排列顺序将有$n!$种. 为便于研究, 引入一个虚拟工件, 其在任何设备上的加工时间均为0, 且在任何一个调度中都放在所有其他工件之前. 对$n+1$个工件$\{J_0, J_1, \cdots, J_n\}$, 定义$D_{i,j}$为相邻两工件的完工时间距离. 根据文献[8], 有

$$D_{i,j}=\begin{cases}\max\limits_{k=1,2,\cdots,m}\left\{\sum\limits_{h=k}^{m}(t_{jh}-t_{ih})+t_{ik}\right\},\\ \qquad i,j=1,2,\cdots,n,\ i\neq j;\\ \sum\limits_{h=1}^{m}t_{jh},\ i=0,\ j=1,2,\cdots,n;\\ \infty,\ i=j\ \text{或}\ j=0.\end{cases}\tag{1}$$

设π为任意一个调度, T为按此调度加工所有工件的最小完工时间, 则T为所有相邻工件在设备上的完工时间距离之和, 即

$$T=\sum_{i=0}^{n-1}D_{i,i+1}.\tag{2}$$

问题的目标是在所有工件的调度集合Π中, 找到π^*, 使得

$$T(\pi^*)\leqslant\ T(\pi),\ \forall\pi\in\Pi.\tag{3}$$

对于给定的调度序列π, 每个工件的开始加工时间S_i的计算公式为

$$S_i=S_{i-1}+D_{0,i-1}+D_{i-1,i}-D_{0,i},\quad i\in\{1,2,\cdots,n\}.\tag{4}$$

3 问题特点及性质

由式(3)发现, 求解n个工件在m台设备上的最小完工时间就是找到一个工件序列, 使得所有相邻工件的完工时间距离之和达到最小.

性质1 根据式(1)和(2)可知, 两相邻工件的完工时间距离与相邻工件在设备上的加工时间有关. 因为工件在每台设备上的加工时间已知, 所以任意两相邻工件间的完工时间距离可预先计算出来. 所有相邻工件的完工时间距离形成一个$(n+1)\times(n+1)$矩阵D(包括虚拟工件), 即

$$D=\begin{bmatrix}\infty & D_{01} & \cdots & D_{0n}\\ D_{10} & \infty & \cdots & D_{1n}\\ \vdots & \vdots & \ddots & \vdots\\ D_{n0} & D_{n1} & \cdots & \infty\end{bmatrix}.$$

因计算D_{ij}的时间复杂度为$O(m)$, 故计算D的时间复杂度为$O(mn^2)$. 任意的D_{ij}都可直接访问矩阵D得到, 因此计算T的时间复杂度降低为$O(n)$.

定理1 对于任意一个调度π, 若将工件i插到工件j之后, 则新的调度π^*的最小化完工时间为

$$T(\pi^*)=T(\pi)-D_{i-1,i}-D_{i,i+1}+D_{i-1,i+1}-D_{j,j+1}+D_{j,i}+D_{i,j+1},\ 0<i,j<n;\tag{5}$$

$$T(\pi^*)=T(\pi)-D_{i-1,i}-D_{j,j+1}+D_{j,i}+D_{i,j+1},\quad i=n,\ 0\leqslant j<n;\tag{6}$$

$$T(\pi^*)=T(\pi)-D_{i-1,i}-D_{i,i+1}+D_{i-1,i+1}+D_{j,i},\quad 0<i<n,\ j=n.\tag{7}$$

证明 当将工件i从调度π中去掉时, 调度π将减少两个相邻工件对$(i-1,i)$和$(i,i+1)$, 同时增加一个相邻工件对$(i-1,i+1)$, 其他工件之间的相邻关系不变, 所以调度π的加工时间增量为$\Delta_1=-D_{i-1,i}-D_{i,i+1}+D_{i-1,i+1}$; 当将工件$i$插到工件$j$之后, 减少了相邻工件对$(j,j+1)$, 同时增加了相邻工件对$(j,i)$和$(i,j+1)$, 所以插入工件$i$对调度$\pi$产生的加工时间增量$\Delta_2=-D_{j,j+1}+D_{j,i}+D_{i,j+1}$, 因此$T(\pi^*)=T(\pi)+\Delta_1+\Delta_2$. 式(5)得证. 同理可证得式(6)和(7). □

定理2 对于任意一个调度π, 任意选取两个工件i和j, 将两工件交换, 形成了调度π^*, 于是新调度的最小化完工时间为

$$T(\pi^*)=T(\pi)-D_{i-1,i}-D_{i,i+1}+D_{i-1,j}+D_{j,i+1}-D_{j-1,j}-D_{j,j+1}+D_{j-1,i}+D_{i,j+1},\ 0<i,j<n,\ i\neq j;\tag{8}$$

$$T(\pi^*)=T(\pi)-D_{i-1,i}+D_{i-1,j}-D_{j-1,j}-D_{j,j+1}+D_{j-1,i}+D_{i,j+1},\quad i=n,\ 0\leqslant j<n;\tag{9}$$

$$T(\pi^*)=T(\pi)-D_{i-1,i}-D_{i,i+1}+D_{i-1,j}+D_{j,i+1}-D_{j-1,j}+D_{j-1,i},\quad j=n,\ 0<i<n-1.\tag{10}$$

证明　在调度 π 中将工件 i 替换成工件 j, 于是调度 π 将减少两个相邻工件对 $(i-1,i)$ 和 $(i,i+1)$, 同时增加两个相邻工件对 $(i-1,j)$ 和 $(j,i+1)$, 因此产生加工时间增量 $\Delta_1 = -D_{i-1,i} - D_{i,i+1} + D_{i-1,j} + D_{j,i+1}$; 同理, 工件 j 被替换成工件 i, 产生的加工时间增量 $\Delta_2 = -D_{j-1,j} - D_{j,j+1} + D_{j-1,i} + D_{i,j+1}$, 因此调度 π^* 相对于调度 π 产生的加工时间增量为 $\Delta = \Delta_1 + \Delta_2$, 即 $T(\pi^*) = T(\pi) + \Delta_1 + \Delta_2$, 式 (8) 得证. 同理可证得式 (9) 和 (10). □

定理 2 同样可应用于两个工件区间的交换, 证明略.

4　邻域迭代搜索算法

邻域迭代搜索算法 (INSA) 的基本思想是: 根据问题特点首先计算相邻工件的完工时间距离; 利用初始解构造算法 (ISC) 构造初始调度序列, 并置为当前解; 对当前解依次执行一遍最佳插入迭代 (BIIA) 和交换迭代 (EIA), 若每次产生的邻域解的完工时间小于当前解, 则取代当前解; 为保证算法的寻优效果, 设置了迭代标志 Flag, 若在一遍迭代过程中, 出现了至少一个新解优于当前解, 则 Flag 为 true, 即需进行下一遍迭代操作, 直至某遍迭代完成时 Flag = Flase, 迭代结束. 当前解即为优化解调度, 根据式 (4) 计算每个工件的加工时间, 算法结束.

INSA 算法具体描述如下:

1) 根据式 (1) 计算相邻工件间的完工时间距离, 即求矩阵 D.

2) 调用初始解构造算法 ISC 生成初始解, 设置搜索标志 Flag = true.

3) 进行邻域迭代搜索. Flag = 最佳插入迭代 BIIA( )or 交换迭代 EIA( ). 如果 Flag = true, 则转 3) 进行下一遍迭代; 否则, 转 4).

4) 根据式 (4) 计算每个工件的开工时间, 输出结果, 算法结束.

4.1　初始解构造 (ISC)

根据无等待流水调度问题的特点, 本问题的解应是所有加工工件构成的一个序列 $\{J_0, J_1, \cdots, J_n\}$, 这里 J_0 为虚设工件, 在调度序列之首. 问题的目标是要找出这样一个序列, 使得总的完工时间最小. 因此设置两个集合 A_1 和 A_2, $A_1 = \{J_0\}$, $A_2 = \{$ 所有待加工工件集合$\}$.

首先在相邻工件完工时间距离矩阵 D 中查找第 0 行中最小值, 假如 $D_{0,i}$ 最小, 则将工件 J_i 加入 A_1 中, 同时从 A_2 中删除工件 J_i. 然后在 D 中查找第 i 行中最小值, 假如最小值为 $D_{i,k}$, 则看工件 J_k 是否已出现在 A_1 中, 如果在, 则说明工件 J_k 已被调度, 需重新在矩阵 D 中查找第 i 行中次最小值, 直到存在一个工件 J_l 不在 A_1 中且为 A_2 中与工件 J_i 的距离最小. 将工件 J_l 从 A_2 中取出放入 A_1 中, 下次从工件 J_l 出发, 查找下一个工件放入 A_1 中, 直到 A_1 中工件个数是 n+1, A_2 变成了空集. 这样所有工件按其进入 A_1 的顺序构成了一个初始解.

算法 ISC 描述如下:

1) $A_1 = \{J_0\}, A_2 = \{J_1, J_2, \cdots, J_n\}, i = 0$;

2) while $(A_2 \neq \phi)$

{在矩阵 D 中从第 i 行中找最小相邻完工时间距离, 若为 $D_{i,k}$ 且 J_k 不包括在 A_1 集合中, 则有

$A_1 \Leftarrow J_k, A_2 \Leftarrow A_2 - J_k, i = k$;

}

3) 将 A_1 中工件按其进入顺序组成初始调度 π^0, 算法结束.

4.2　最佳插入迭代算法 (BIIA)

置初始调度 π^0 为当前调度 π, 进行最佳插入迭代. 基本思想是: 对 π 中的每个工件依次进行如下操作: 首先将其从调度序列 π 中删除, 然后分别尝试插入剩余的 $N-1$ 个可能的位置 (注: 不能插在 J_0 之前). 根据定理 1 分别计算其插入到其他工件后的完工时间增量 Δ, 并找出最小的 Δ. 如果 Δ 小于零, 则找到一个比当前调度 π 更好的调度 π^*, 并将该工件插入到产生最小 Δ 的位置上, 置 π^* 为当前调度 π; 否则当前调度不变. 重复上述过程, 直至对所有的节点都进行了如上操作, 于是完成了一遍最佳插入迭代.

BIIA 算法具体描述如下:

bool BIIA ( )

{$i = 1$; Flag = false;

while $(i \leqslant n)$

{min = 100; $k = -1$;

for $(j = 0; j \leqslant n; j++)$

{计算将 π 中第 i 个工件插入第 j 个工件后的增量 Δ_j;

if $(\min > \Delta_j)$ min = Δ_j; $k = j$;

}

if $(\min < 0)$

则将 π 中第 i 个任务插入最小 Δ 产生的位置 k 上, 形成 π^*;

$\pi \Leftarrow \pi^*$; Flag = true;

else

$i = i + 1$; //有些工件可能插入多次

}

return Flag.}

4.3 交换迭代算法 (EIA)

为了最大可能地得到最优解, 避免最佳插入迭代陷入局部最优, 在上一节插入迭代的基础上, 对获得的调度π进一步交换迭代. 基本思想是: 随机产生两个$1 \sim n$的数R_1和R_2作为两个交换区间的起始点, H代表交换区间长度, 值从1变化到$\sqrt{n}$. 将这两段工件序列区间进行交换, 产生交换增量Δ, 根据定理2来计算Δ. 如果Δ小于0, 则将两个工件区间进行交换后形成新调度π^*, 令$\pi \Leftarrow \pi^*$, 重复上述过程, 直到连续K次交换都不能改进π, 算法停止, 返回当前解.

EIA算法描述如下:

bool EIA ( )

{Flag = false;

for $(H = 1$ to $\sqrt{n})$

$\{K = 0$;

while $(K < 20)$

{产生两个取值在$1 \sim n$之间的随机数R_1和R_2, 令$R_1 < R_2$, 否则交换R_1和R_2的值;

if $R_1 + H \geqslant R_2$

则说明两个交换区间有叠加, 于是重新产生随机数R_1和R_2, 直至产生的区间不叠加;

else

计算交换区间$R_1 \sim R_1 + H$和$R_2 \sim R_2 + H$上的工件所产生的Δ. 若$R_2 + H > n$, 则计算交换区间$R_1 \sim R_1 + h$和$R_2 \sim n$上的工件所产生的Δ.

if $\Delta < 0$

则交换对应区间上的工件得到调度π^*,

令$\pi \Leftarrow \pi^*; K = 0;$Flag = true;

else

$K = K + 1$;

}

}

return Flag.}

上述算法中K为设置的一个交换阈值, 意思为如果连续的K次任意交换都不能对调度π有所改进, 则停止对区间长度为H的交换, 并开始区间长度为$H + 1$的交换. 经反复实验确定$K = 20$. H是一个区间长度变量, 在此用来调整交换区间的大小, 通过加大交换区间来增加解跳出局部最优的能力, 从而保证找到全局最优解.

5 仿真实验及分析

为了说明算法的执行流程, 以7个工件在5台机器上的无等待流水调度问题为例, 相应的加工时间如表1所示.

表1 7个工件在5台机器上的加工时间表

机器	工 件						
	J_1	J_2	J_3	J_4	J_5	J_6	J_7
M_1	41	53	28	58	28	73	21
M_2	65	40	26	42	53	75	71
M_3	39	8	83	67	20	35	31
M_4	12	3	99	88	9	85	70
M_5	4	24	58	72	57	90	90

Step 1: 计算任意两相邻工件的完工时间距离矩阵D, 即

$$D = \begin{bmatrix} \infty & 161 & 128 & 294 & 327 & 167 & 358 & 283 \\ \infty & \infty & 24 & 224 & 214 & 84 & 238 & 207 \\ \infty & 86 & \infty & 231 & 252 & 104 & 283 & 227 \\ \infty & 4 & 24 & \infty & 102 & 57 & 117 & 102 \\ \infty & 4 & 24 & 85 & \infty & 57 & 103 & 90 \\ \infty & 34 & 24 & 180 & 188 & \infty & 219 & 176 \\ \infty & 4 & 24 & 67 & 72 & 57 & \infty & 90 \\ \infty & 4 & 24 & 80 & 78 & 57 & 96 & \infty \end{bmatrix},$$

D为一个8×8矩阵.

Step 2: 调用ISC算法产生初始调度$\pi = \{J_0, J_2, J_1, J_5, J_7, J_4, J_3, J_6\}$, 其加工时间为754.

Step 3: 利用BIIA算法对调度π进行改善, 经改善后的调度$\pi^* = \{J_0, J_5, J_7, J_4, J_6, J_3, J_1, J_2\}$, 新调度的生产时间为619, Flag = true, $\pi \Leftarrow \pi^*$.

Step 4: 利用EIA算法对调度π进一步改进, 结果为π不变, 此时Flag仍为true. 因此需要进行第2遍迭代, 即再次执行BIIA算法和EIA算法. 因为第2遍迭代中调度π没有被改善, 所以Flag为false, 退出迭代, 找到优化调度π.

Step 5: 求解每个作业在第1台机器上的开始加工时间, 分别为:

J_5为0时, J_7为60时, J_4为94时, J_6为166时, J_3为297时, J_1为434时, J_2为491时.

为了评价INSA算法的性能, 采用14个典型调度问题及2组随机问题作为测试数据, 在处理器为P (R52) M1.86 GHz, 内存512 MB的PC机上进行仿真实验. 每个算例独立运行10次, 所得计算结果如表2所示. 其中: 偏差是指相对于算法RAJ所得解的偏差, 时间是指在相同机器上算法运行的时间.

表 2　相关算法所得结果

调度名称	问题规模 $M \times N$	RAJ 算法最优完工时间	TS+M 算法		DPSO 算法		INSA 算法		
			平均偏差	时间	平均偏差	时间	优化完成时间	时间	均方差
Rec 01	5× 20	1 561	−3.12	0.57	−3.69	0.00	1 557	0.00	0.00
Rec 03	5× 20	1 426	−5.93	0.568	−6.04	0.01	1 421	0.00	0.00
Rec 07	10× 20	2 078	−3.32	1.46	−3.63	0.01	2 074	0.00	0.00
Rec 09	10× 20	2 134	−4.73	1.59	−5.14	0.02	2 130	0.01	0.00
Rec 13	15× 20	2 688	−6.05	2.11	−6.05	0.08	2 683	0.01	0.00
Rec 15	15× 20	2 660	−6.02	2.04	−6.02	0.09	2 653	0.00	0.00
Rec 21	10× 30	2 951	−6.75	2.12	−6.88	0.16	2 945	0.02	0.00
Rec 23	10× 30	2 902	−11.73	2.20	−10.89	0.15	2 893	0.00	0.00
Rec 27	15× 30	3 601	−5.98	3.50	−6.14	0.43	3 594	0.03	0.01
Rec 29	15× 30	3 391	−8.34	3.39	−8.23	0.43	3 380	0.02	0.11
Rec 31	10× 50	4 631	−6.04	5.98	−6.15	0.82	4 622	0.05	0.04
Rec 33	10× 50	4 746	−6.46	6.10	−6.78	0.89	4 740	0.05	0.08
Rec 37	20× 75	8 471	−9.63	12.56	−10.52	1.77	8 461	0.09	0.15
Rec 39	20× 75	8 867	−7.05	12.79	−7.69	1.89	8 860	0.11	0.23
随机 01	20× 200	2 043	−11.23	523.76	−10.33	64.23	2 037	1.001	0.05
随机 02	20× 500	4 821	−23.16	9 045.35	−23.43	275.19	4 804	2.039	0.12

由表 2 可得如下结论: 在算法运行效率方面, 当问题规模较小时, INSA 算法与 DPSO 运行时间基本一样, 而 TS+M 算法运行时间稍长, 但影响不大; 随着问题规模的变大, TS+M 算法的运行时间明显增大, DPSO 次之, 而 INSA 算法的运行时间并没有明显增大; 特别在问题规模为 20×500 时, INSA 算法需要运行 2.039 s, DPSO 算法需要运行 275.19 s, 而 TS+M 算法需要运行 9 045.35 s, 大约为 2.5 h. 这说明随着问题规模的变大, DPSO 和 TS+M 算法的效率急剧下降, 其根本原因是由于通过大幅度扩大搜索范围来提高解的质量所造成的. 这种长的运行时间与现场快速的生产调度要求相矛盾. INSA 算法由于采用了增量计算, 在求解邻域解时算法的时间复杂度变为 0, 从而极大地提高了算法运行效率. 当问题规模为 20×500 时, INSA 算法仅 2 s 多, 所以调度结果令人满意, 非常适合现场调度的需要.

从算法性能角度来看, INSA 算法得出的解与 TS+M 和 DPSO 算法得出的解相比较, 部分寻优结果稍差, 且差距不大, 而部分解的优化性能比 TS+M 和 DPSO 的解优化度高; 另外, 利用 INSA 算法求解同一问题 10 次的均方差都很小, 因此 INSA 算法的鲁棒性很好.

6　结　论

本文在研究最小化最长完工时间的无等待流水作业调度问题的基础上, 通过分析问题特征, 总结了插入操作对调度序列最小化完工时间的影响规律 (如式 (5) ∼ (7)) 和交换操作的影响规律 (如式 (8) ∼ (10)), 提出了基于邻域迭代的搜索算法. 该算法利用相邻工件间的完工时间距离求解 Makespan, 使计算 Makespan 的时间复杂度降了一阶; 同时在最佳插入迭代算法 BIIA 和交换迭代算法 EIA 中利用式 (5) ∼ (10) 中的增量法计算 Makespan, 又将算法的时间复杂度降了一阶. 仿真实验表明, INSA 算法在大规模问题上的效率明显优于 DPSO 和 TS+M; 在性能上与 DPSO 和 TS+M 基本相当, 且鲁棒性较好. 因此, 该算法能很好地适应于大规模调度以及动态调度等问题.

参考文献(References)

[1] Van Deman J M, Baker K R. Minimizing mean flow time in the flow shop with no intermediate queues[J]. AIIE Transactions, 1974, 6(1): 28-34.

[2] Gangadharan R, Rajendran C. Heuristic algorithms for scheduling in the no-wait flowshop[J]. Int J of Production Economics, 1993, 32(3): 285-290.

[3] Rajendran C. A no-wait flowshop scheduling heuristic to minimize makespan[J]. J of the Operational Research Society, 1994, 45(4): 472-478.

[4] Bonney M C, Gundry S W. Solutions to the constrained flow-shop sequencing problem[J]. Operational Research Quart, 1976, 24(4): 869-883.

[5] King J R, Spachis A S. Heuristics for flow-shop scheduling[J]. Int J of Products Research, 1980, 18(3): 343-357.

[6] 潘全科, 赵保华, 屈玉贵. 无等待流水车间调度问题的优化[J]. 计算机学报, 2008, 31(7): 1147-1154.
(Pan Q K, Zhao B H, Qu Y G. Heuristics for the no-wait flow shop problem with Makespan criterion[J]. Chinese J of Computers, 2008, 31(7): 1147-1154.)

[7] Grabowski J, Pempera J. Some local search algorithms for no-wait flow shop problem with makespan criterion[J]. Computers and Operations Research, 2005, 32(8): 2197-2122.

(下转第547页)