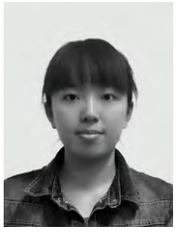
文章编号：1671-7848(2018)02-0362-05 **DOI:**10.14107/j.cnki.kzgc.160321

基于变邻域搜索的可重入流水车间调度

王丹敬,徐建有
(东北大学 信息科学与工程学院，沈阳 110819)



摘 要：与传统的流水车间调度问题中工件在每个机器上只能加工一次不同，考虑了机械制造中工件需要在一个机器上进行多次重复加工的实际情况，研究了工件可重入的流水车间调度问题。针对该问题，基于对邻域搜索性能的分析和学习，提出了一个自适应变邻域搜索算法，并在算法中嵌入了一个精英解集合，以增强算法的跳出局部最优的能力。基于随机测试问题的实验结果表明，所提出的自适应策略能够明显增强变邻域搜索算法的搜索效率，使得算法能够快速获得高质量的近优解，并且其性能要优于 CPLEX 优化软件。

关键词号：TP18 **文献标识码**：A

A Variable Neighborhood Search for the Re-entrant Permutation Flowshop Scheduling Problem

WANG Dan-jing, XU Jian-you
(College of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: Different from the traditional permutation flowshop scheduling in which each job can be processed on each machine for at most once, in practical production of the mechanical industry a job generally needs to be processed on a machine for several times. So this paper considers this practical condition and investigates the re-entrant permutation flowshop scheduling problem. For this problem, an adaptive variable neighborhood search (AVNS) algorithm is proposed based on the analysis and learning of search performance of each neighborhood. In addition, an elite solution set is embedded in the algorithm so as to enhance its ability of getting out of local optimum. Computational results based on randomly generated instances show that the proposed adaptive strategy can significantly enhance the search efficiency so that the AVNS can achieve a high quality near-optimal solution very quickly. On average, the proposed AVNS is also superior to the software CPLEX.

Key words: Re-entrant job; permutation flowshop scheduling; variable neighborhood search

1 引 言

在传统的流水车间调度(Permutation Flowshop Scheduling, PFS)问题中，通常都假设每个工件只能在每台机器上加工一次。但是，这种假设在机械制造等一些实际工业生产过程中却不成立，因为在这类工业中，很常见的一种现象是工件需要反复多次在同一台机器上进行加工。这种车间在文献中通常称为“可重入车间”，即一个工件可以在机器上反复进行多次加工[1]。这种现象在其他工业中也比较常见，例如模具热处理生产过程[2]、半导体晶片的生产过程[3]、信号处理[4]、印刷电路板[5-6]等。

近年来，带有工件可重入的生产调度问题已经受到许多学者的关注。但是，如何有效求解这类问题仍然是一个具有挑战性的课题，因为工件的可重入会导致问题结构变得非常复杂，以及许多需要关注的问题：如避免死锁、能力分配、库存控制、工件调度等。而以最小化 makespan (工件的最大完成时间)为目标的可重入排列排序流水车间调度(Re-entrant PFS, RPFS)问题则是强 NP-难的[7]。

针对该问题，Pan 和 Chen[8]以最小化 makespan 为目标，针对大规模问题提出了 6 个启发式算法。Alfieri[9]研究了一个从纸板工业中提炼出来的一个多目标可重入流水车间调度问题。Liu[10]针对以最小化工件的总加权拖期的 RPFS 问题，提出了一个遗传算法。Rau 和 Cho[11]针对一个可重入生产系统的检查分配问题提出了一个遗传算法。Lin 和 Lee[12]提出了一个遗传算法，在算法中的编码使用了一个多层染色体，以反映可重入的机率和资源消耗之间的依赖关系。

收稿时间：2016-03-02 **修回时间**：2016-04-15
作者简介：王丹敬(1997–)，女，山东滕州人，本科生，主要研究方向为生产调度与智能优化算法等。

针对不同生产环境中的 RPFS 问题，Choi 等[13]研究了一个包含 2 个阶段的可重入混合流水车间调度问题。Chu 等[14]考虑了一个由两台机器构成的 RPFS 问题。刘勇等[15]针对存在批处理机的可重入混合复杂流水线生产调度问题提出了一个基于约束规划的调度算法。

2 问题描述

RPFS 问题可以描述如下：有 n 个工件要在 m 个机器 $M_1, \ldots, M_m$ 上进行加工；假设工件可能需要在一些机器上进行 L 次重复加工(每次加工称为一级，这样每个工件就有 L 级加工)，并且 $L>1$；工件在 m 个机器上的加工顺序必须相同且按照 $M_1, M_2, \ldots, M_m$ 的顺序进行；所有工件在每一级加工中的顺序都相同，即每个工件都要按照 $M_1, M_2, \ldots, M_m$ 的顺序重复进行 L 次加工，并且一旦工件的排序确定之后，那么在每级加工时工件的排序都不会改变，即工件 1 至工件 n-m 在完成一级加工后重新回到机器 M_1 的前面等待开始下一级的加工；问题的优化目标是最小化 makespan。

3 自适应变邻域搜索算法 AVNS

3.1 传统的变邻域搜索算法

由于一个邻域内的局部最优解未必是其它邻域的局部最优解，因此，VNS 算法的基本思想是通过一定的规则，使得算法能够在不同的邻域之间进行转换，从而使得算法能够具有较好的跳出局部最优解的能力。传统 VNS 算法的流程，如算法 1 所示。

算法 1：传统的 VNS 算法

```
1  输入：初始解 s，邻域 N_d (d=1, …, K)
2  while 未达到停止准则
3     d : =1
4     while d ≤ K do
5        s′: = Shaking (s, N_d)
6        s″: = LocalSearch (s′, N_d)
7        if f(s″) < f(s)
8           s: = s″
9           d : =1
10       else
11          d : = d+1
12       end if
13    end while
14 end while
```

在该算法中，输入项为一个初始解和一个邻域集合(由 K 个邻域构成)。在 VNS 算法的每一次迭代过程中，算法从邻域 d=1 开始搜索，首先，使用 *Shaking* 函数对解 s 进行一次扰动(通常是从邻域 N_d 内随机产生一个移动，再将其应用到解 s 上)，得到一个扰动解 s'；然后，对 s'在邻域 N_d 内进行邻域搜索，得到其局部最优解 s''，如果该解的目标函数 $f(s'')$ 比解 s 差，则转到下一个邻域(即令 d=d+1)；否则，就将 s''赋给 s，并再次从第一个邻域开始搜索。

3.2 AVNS 算法

从传统 VNS 算法流程可以看出，该算法具有结构简单并且搜索效率较高的特点，但是，该算法也存在不足之处：首先，邻域的搜索顺序通常是根据经验进行设定，一旦找到更好的解，算法就会重新从第一个邻域开始搜索。如果针对当前问题，前面的邻域搜索效果不好，算法就会浪费大量的时间进行无效搜索。其次，算法在每次迭代中都是从当前最好解 s 开始搜索，如果该解已经陷入局部最优而当前所使用的邻域类型的广域搜索能力有限，那么就会出现算法长期陷入局部最优的现象。因此，针对以上两个问题，本文提出了以下改进策略，从而构成了一个新的自适应 VNS 算法。

① 将每种邻域的搜索效果进行评价，利用其索效果提出一个邻域的自适应选择策略，即邻域的变换不是按照其预定的邻域顺序，而是按照邻域搜索的性能，使得算法能够自适应地选择那些针对当前问题搜索效果较好的邻域类型，从而提高进一步提高算法的搜索效率；

② 在算法中引入一个精英解集合策略，即将算法搜索过程中所找到的质量较好的解存储在一个精英解集合中，算法在每次迭代时，从这个精英解集合中随机选择一个解作为搜索的起点，从而防止算法出现长时间陷入局部的现象。

3.2.1 解的编码与初始解生成

在所提出的 AVNS 算法中，使用所有的 n 个工件的一个排序来表示一个解(对应于一个生产调度方案)，即解 $S=(s(1), s(2), \ldots, s(i)\ldots, s(n))$，其中, $s(i)$ 表示被安排在解 S 的第 i 个位置的工件号。

第一个解使用 NEH 方法来产生[17]，因为该方法被证明是产生 PFS 问题的性能最好的简单启发式方法，因而可以保证初始解具有较好的质量。该方法主要包括以下几个过程：

Step 1　将所有的工件根据其在所有机器的总处理时间按照从小到大的顺序排列 P；

Step 2　在所得到的工件排序中，先确定前 2 个工件的最优排序，得到一个部分解的排序 S'，再将这 2 个工件从 P 中删除；

Step 3　以当前所得到的部分解 S'为基础，从剩余的工件排序中取第一个位置的工件，再将其插入到 S'中能够使得目标函数的增加量最小的位置上，然后将该工件从 P 中删除；

Step 4　可以重复进行 Step 3，直到 P 变为空集，输出所得到的完整解 S'。

3.2.2 邻域类型

在所提出的 AVNS 算法中，使用所有的 n 个工件的一个排序来表示一个解(对应于一个生产调度方案)，即解 $S=(s(1), s(2), \ldots, s(i)\ldots, s(n))$，其中，$s(i)$ 表示被安排在解 S 的第 i 个位置的工件号。

在 AVNS 算法中，主要使用 3 种邻域类型，其中前两种邻域类型在 PFS 问题中经常采用。

① *insertion* 邻域：该邻域基于 *insertion* 移动，即从当前解中删除一个工件，并把它依次插入到解中的其它位置上。通常的做法是对于解中的每个工件，都执行这样的 *insertion* 移动，然后取一个最好的解作为该邻域内的最好解。该邻域的计算复杂度为 $O(n(n-1))=O(n^2)$。

② *swap* 邻域：该邻域基于 *swap* 移动，即选定一个工件，将其与其它的工件进行交换，对所有工件均执行这样的 *swap* 移动，然后取一个最好的解作为该邻域的最好解。该邻域的计算复杂度为 $O(n(n-1)/2)= O(n^2)$。

③ 2-*insertion* 邻域：与 *insertion* 邻域移动不同，2-*insertion* 邻域移动首先从当前解中删除两个相邻的工件，然后按照它们删除的顺序，再将它们依次插入到当前解中最好的位置上(即能够使目标函数值增加量最小的位置)。该邻域的计算复杂度为 $O((n-1)(n-1)n)=O(n^3)$。

从以上的描述可以看出，2-*insertion* 邻域相比于 *insertion* 邻域，其邻域的规模要更大，从而使得获得更好解的机率也变大。

3.2.3 自适应邻域选择策略

如前所述，该自适应策略的目的是通过对各邻域搜索结果的分析，建立各邻域类型的选择概率，从而使得算法能够自适应地选择更加适合当前所求解问题的邻域类型。为此，算法按照如下原则来定义邻域的选择概率：如果一个邻域类型被采用后，它所获得的局部最优解优于所输入的解，则认为该邻域类型成功一次，否则认为其失败一次。这样，在算法中就可以统计各邻域类型 i 的成功和失败次数，分别定义为 s_i 和 f_i，那么该邻域类型的成功率 $SR_i = s_i/(s_i+f_i)+\varepsilon$。成功率最后加上一个非常小的正数 ε ($\varepsilon < 0.1$)，是为了防止出现某个邻域类型一直没有找到更好解而导致其成功率为 0 的情况。基于各邻域类型的成功率，其选择概率定义为 $p_i = SR_i \Big/ \sum_{k=1}^{3} SR_k$。在确定了选择概率之后，使用遗传算法中常用的轮盘赌方法选择邻域类型。

3.2.4 AVNS 算法流程

本文提出的 AVNS 算法的主要流程，见算法 2。

算法 2：AVNS 算法

输入：初始解 S，邻域 N_d (d=1, 2, 3)
初始化各邻域的选择概率为 p_i=1/3，成功次数与失败次数 $s_i = f_i = 0$，令精英解集合 $E=\{S\}$
while 未达到停止准则
　从精英解集合 E 中随机取一个解，作为 S
　基于各邻域的选择概率，使用轮盘法随机选择一个邻域类型，假设为 N_k
　S': = *Shaking* (S, N_k)
　S'': = *LocalSearch* (S', N_k)
　if $f(S'') < f(S)$
　　S: = S''　//替换 E 中相应的解
　　s_i : = s_i +1
　else
　　f_i : = f_i +1
　　if 精英解集合 E 中解的数量未超限 **do**
　　将解 S''加入到 E 中
　　else if 解 S''优于 E 中最差的解 **do**
　　用解 S''替换 E 中最差的解
　　end if
　end if
　更新各邻域类型的选择概率 p_i
end while
输出 E 中最好的解作为最终解

在算法中，精英解集合 E 的大小设计为 10。在由邻域搜索所得到的局部最优解 S''比输入的解 S 的质量差时。

如果 E 中解的数量未超限，则直接将其加入到 E 中；否则判断其质量是否优于 E 中最差解，如果是，则用其替换 E 中最差解。

4 仿真实验

4.1 实验设置

在算法实验中，本文所提出的 AVNS 算法使用 C++语言编写，并在 Intel Core2 Q9550 (2.83 GHz)的 CPU 和 4 GB 内存的个人计算机上进行了测试。

在实验中，本文使用 Xu *et al.*[18]所提出的测试问题。在这些测试问题中，工件的处理时间在[1, 100]之间随机产生，工件的数量从 5 到 30，机器的数量从 5 到 20，重入次数从 3 到 10 变化。

这些测试问题共包含 330 个算例，此外，算法的停止准则为最大允许计算时间达到 $0.1\times n\times m\times L$ s，其中，n 为工件数量、m 是机器数量、L 为工件的重入次数。针对这些算例，Xu *et al.*[18]使用 CPLEX 对其进行求解，以获得该算例的最优解或者它的一个下界。

4.2 实验结果与分析

对于重入次数 L=3 和 L=5，所有的 12 组规模都进行了测试，对于 L=10，只有 9 个规模进行了测试，因为对于该规模的问题，CPLEX 无法求解其中的 3 组。

在实验中，将所提出的 AVNS 算法与使用 3.2.2 节三种邻域的传统 VNS 算法以及 Xu et al.[18]所提出的 Memetic 算法(记为 MA)进行了比较，在实验中将 VNS 以及 AVNS 的最大运行时间设定为 MA 算法的实际运行时间。

各算法的计算结果，见表 1~表 5。

表 1 L=3 时的各算法性能比较
Tab. 1 Comparison of different algorithms for L=3

Problem	Gap(%)				CPU(s)	
(n×m×L)	*CPLEX*	*VNS*	*MA*	*AVNS*	*CPLEX*	*AVNS*
5×5×3	**0.00**	0.65	0.65	0.65	0.12	2.25
5×10×3	**0.00**	**0.00**	**0.00**	**0.00**	0.36	0.00
5×20×3	**0.00**	0.05	**0.00**	**0.00**	0.48	0.01
10×5×3	**0.00**	0.71	0.66	0.60	2.3	9.00
10×10×3	**0.00**	0.16	0.13	0.10	251.45	6.01
10×20×3	**0.00**	0.07	**0.00**	**0.00**	334.43	0.02
20×5×3	**0.31**	1.32	1.18	1.07	543.38	15.33
20×10×3	3.39	1.87	1.56	**1.47**	2 709.34	60.00
20×20×3	16.69	13.09	12.37	**12.05**	5741.55	120.01
30×5×3	0.12	0.11	0.09	**0.07**	966.95	16.55
30×10×3	4.67	2.39	2.00	**1.83**	2 846.92	90.02
30×20×3	11.10	7.61	6.84	**6.61**	3 166.24	180.03
Average	3.02	2.34	2.12	**2.04**	1 380.29	41.60

表 2 L=5 时的各算法性能比较
Tab.2 Comparison of different algorithms for L=5

Problem	*Gap*(%)				*CPU*(second)	
(n×m×L)	*CPLEX*	*VNS*	*MA*	*AVNS*	*CPLEX*	*AVNS*
5×5×5	**0.00**	0.39	0.39	0.39	0.23	6.25
5×10×5	**0.00**	**0.00**	**0.00**	**0.00**	0.52	0.00
5×20×5	**0.00**	0.02	**0.00**	**0.00**	1.89	0.01
10×5×5	**0.00**	0.84	0.84	0.80	23.44	17.51
10×10×5	4.51	3.93	3.91	**3.85**	762.9	50.00
10×20×5	**0.00**	0.03	0.03	0.03	1446.29	10.15
20×5×5	**0.00**	0.73	0.70	0.70	764.49	30.21
20×10×5	3.17	1.75	1.69	**1.62**	2 356.66	100.01
20×20×5	19.59	15.62	15.55	**14.90**	5 241.10	200.03
30×5×5	**0.16**	1.01	0.97	0.85	1 175.2	60.05
30×10×5	3.17	1.63	1.51	**1.44**	2 326.1	150.02
30×20×5	10.47	7.04	6.64	**6.48**	4 463.32	300.06
Average	3.42	2.75	2.69	**2.59**	1 546.83	77.03

续表

表 3 L=10 时的各算法性能比较
Tab.3 Comparison of different algorithms for L=10

Problem	Gap(%)				*CPU*(s)	
(n×m×L)	*CPLEX*	*VNS*	*MA*	*AVNS*	*CPLEX*	*AVNS*
5×5×10	**0.00**	0.40	0.40	0.40	0.66	15.00
10×5×10	**0.01**	1.51	1.51	1.51	314.12	50.00
10×10×10	2.41	2.35	**2.31**	**2.31**	2402.47	90.02
10×20×10	4.92	4.50	4.43	**4.38**	2 620.85	200.01
20×5×10	**0.15**	0.73	0.70	0.65	1 853.29	80.49
20×10×10	4.20	2.78	2.74	**2.61**	2 422.46	200.03
20×20×10	21.64	18.05	17.90	**17.62**	5 269.87	400.05
30×5×10	**0.23**	0.95	0.92	0.86	1 401.81	105.09
30×10×10	2.61	1.35	1.28	**1.09**	2 457.5	300.05
Average	4.02	3.62	3.58	**3.49**	2 082.56	160.08

表 4 各算法性能随工件数变化统计结果
Tab. 4 Performance of different algorithms as n increases

n	*Gap*(%)			
	CPLEX	*VNS*	*MA*	*AVNS*
5	**0.000 0**	0.215 7	0.205 7	0.205 7
10	**1.316 7**	1.566 7	1.535 6	1.508 9
20	7.682 2	6.215 6	6.043 3	**5.854 4**
30	4.066 3	2.761 3	2.531 3	**2.403 8**
Average	3.266 3	2.689 8	2.579 0	**2.493 2**

表 5 各算法性能随机器数变化统计结果
Tab.5 Performance of different algorithms as m increases

n	Gap(%)			
	CPLEX	*VNS*	*MA*	*AVNS*
5	**0.081 7**	0.779 2	0.750 8	0.712 5
10	2.557 3	1.655 5	1.557 3	**1.483 6**
20	8.441 0	6.608 0	6.376 0	**6.207 0**
Average	3.693 3	3.014 2	2.894 7	**2.801 0**

表中 CPLEX 与 MA 算法的结果为文献[18]所报道的算法结果，其中 CPLEX 的结果为其所得到的上界与下界之间的平均偏差。从结果中可以得出：

① 随着问题规模(n、m、L)的增大，各算法所

获得最好解的偏差值和求解时间都不断增大。在求解质量上，VNS、MA 和 AVNS 的性能在小规模时都要比 CPLEX 差。但是针对 CPLEX 无法求得最优解的中等和大规模问题，这 3 个算法的性能都要优于 CPLEX。

② MA 算法的性能要优于传统的 VNS 算法，而本文所提出的 AVNS 算法的性能均要明显优于传统的 VNS 算法和文献[18]中的 MA 算法。这充分表明了本文所提出的 AVNS 算法的有效性。

AVNS 算法相对于其它算法能够取得更好结果的原因主要在于以下两点：

① AVNS 算法不需要像 MA 算法那样去维护一个种群，而只是从一个精英解集中去选择一个解来进行搜索，从而在保证搜索分散性的同时节省了一定的搜索时间，算法能够进行更多次的迭代搜索；

② AVNS 算法能够自适应地选择最适合当前问题的邻域类型，从而可以提高搜索效率。

5 结　论

针对机械制造工业中工件通常需要在机器上进行重复多次加工的实际情况，本文研究了工件具有可重入情况的流水车间调度问题。基于对传统变邻域 VNS 算法的优势及其在生产调度问题中的成功应用，在对其不足之处进行分析的基础上，提出了一个自适应 VNS 算法。该算法的显著特点是采用的多个不同邻域之间没有固定的搜索顺序，在搜索过程中是根据它们的搜索效果，由算法自适应地去选择，从而能够保证针对当前问题更加有效的邻域类型能够获得较多的采用次数，从而进一步提高 VNS 算法的搜索效率。基于当前文献中已有的标准测试问题的实验结果表明，所提出的自适应 VNS 算法要明显优于传统的 VNS 算法，从而证明了所提出的自适应策略的有效性。此外，所提出的自适应 VNS 算法的性能也要优于当前文献中已经有的 memetic 算法，从而进一步表明了本文所提出的自适应 VNS 算法的有效性。

参考文献(References)

[1] 王利存, 郑应平. 连续时间可重入生产系统调度策略灵敏度分析[J], 计算机集成制造系统, 2001, 7(2): 10-14.
Wang L C, Zheng Y P. Sensitivity Analysis of Continuous Time Re-entrant Production System Scheduling Policies [J]. 2001, 7(2): 10-14.

[2] 林刚, 刘建军, 陈庆新, 等. 可重入 flow-shop 类型模具热处理车间动态批调度[J], 计算机集成制造系统, 2016, 4(22): 1046-1058.
Lin G, Liu J J, Chen Q X, etal. Dynamics Scheduling for the Re-entrant Mould Heat-treatment Flow-shop [J], Computer Integrated Manufacturing Systems, 2016, 4(22): 1046-1058.

[3] Vargas-Villamil FD, Rivera DE. A model predictive control approach for real-time optimization of reentrant manufacturing lines[J]. Computers in Industry, 2001, 45(2): 45-57.

[4] Wang M Y, Sethi SP, Velde S L. Minimizing Makespan in a Class of Reentrant Shops [J], Operations Research, 1997, 45(2): 702-712.

[5] Beng G. A simulation-based scheduler for flexible flowlines[J], International Journal of Production Research, 1994, 32: 321-344.

[6] Bispo C F, Tayur R. Managing Simple Re-entrant Flow Lines: Theoretical Foundation and Experimental Results [J], IIE Transaction, 2001, 33(3): 609-623.

[7] Bispo CF, Tayur R. Managing Simple Re-entrant Flow Lines: Theoretical Foundation and Experimental Results [J], IIE Transaction, 2001, 33: 609-623.

[8] Pan J C H, Chen J S. Minimizing Makespan in Re-entrant Permutation Flow-shops [J], Journal of the Operational Research Society, 2003, 54(3): 642-653.

[9] Alfieri A. Workload Simulation and Optimisation in Multi-criteria Hybrid Flowshop Scheduling: A case study [J], International Journal of Production Research, 2009, 47(18): 5129-5145.

[10] Liu C H. A Genetic Algorithm Based Approach for Scheduling of Jobs Containing Multiple Orders in a Three–machine Flowshop [J], International Journal of Production Research, 2010, 48(15): 4379-4396.

[11] Rau H, Cho K H. Genetic Algorithm Modeling for the Inspection Allocation in Reentrant Production Systems [J], Expert Systems with Applications, 2009, 35(8): 11287-11295.

[12] Lin D, Lee C K M. A multi-level GA Search with Application to the Resource-constrained Re-entrant Flow Shop Scheduling Problem [J], World Academy of Science, Engineering and Technology, 2012, 64(4): 746-750.

[13] Choi H S, Kim H W, Lee D H, etal. Scheduling Algorithms for Two-stage Reentrant Hybrid Flow Shops: Minimizing Makespan under the Maximum Allowable Due Dates [J], International Journal of Advanced Manufacturing Technology, 2009, 42(3): 963-973.

[14] Chu F, Chu C, Desprez C. Series production in a basic re-entrant shop to minimize makespan or total flow time [J], Computers & Industrial Engineering, 2010, 58: 257-268.

[15] 刘勇, 谷寒雨, 席裕庚. 基于约束理论的混合复杂流水线规划调度算法[J], 计算机集成制造系统, 2005, 11(1): 97-103.
Liu Y, Gu H Y, Xi Y G. Planning and Scheduling Algorithm Based on TOC for Complex Hybrid Flowshop Problems [J], 2005, 11(1): 97-103.

[16] Hansen P, Mladenovic N, Moreno P, Jose A. Variable Neighborhood Search: Methods and Applications [J], Annals of Operations Research, 2010, 175(1): 367-407.

[17] Nawaz M, Enscore EE, Ham I. A Heuristic Algorithm for the m-machine, n-job Flow Shop Sequencing Problem [J].OMEGA, 1983, 11(2): 91-95.

[18] Xu J Y, Yin Y Q, Cheng TCE, etal. A Memetic Algorithm for the Re-entrant Permutation Flowshop Scheduling Problem to Minimize the Makespan[J], Applied Soft Computing, 2014, 24(3), 277-283.