文章编号: 1001-0920 (2012) 12-1781-06

基于新邻域结构的 Memetic 算法求解流水车间调度问题

徐建有, 顾树生

(东北大学 信息科学与工程学院，沈阳 110819)

摘 要: 流水车间调度是一类典型的生产调度问题, 属于 NP-难问题. 针对传统的最优化方法难以求解大规模问题, 提出了一个 Memetic 算法, 在算法的局部搜索中使用一种新型的基于 NEH 的邻域结构, 并且其邻域规模随着搜索的进行能够动态变化, 可以大大提高算法的搜索能力. 通过对标准 Benchmark 问题的测试, 所得结果表明提出的基于新邻域结构的 Memetic 算法具有较好的性能, 并且优于已有文献中的粒子群算法.

关键词: 流水车间调度；NEH 邻域；Memetic 算法

中图分类号: TP273 **文献标志码:** A

New neighbourhood based Memetic algorithm for permutation flowshop scheduling problem

XU Jian-you, GU Shu-sheng

(College of Information Science and Engineering，Northeastern University，Shenyang 110819，China. Correspondent: XU Jian-you，E-mail：xujianyou@ise.neu.edu.cn)

Abstract: The permutation flowshop scheduling problem(PFSP) is a kind of classical production scheduling problem, which is N_P-hard. The traditional optimization method cannot be adopted to solve large scale problems. Therefore, this paper proposes a Memetic algorithm for the PFSP, in which a new neighbourhood structure based on NEH is developed. In addition, the size of the neighbourhood is dynamically adjusted during the search process. The computational results on Benchmark problems show that the proposed Memetic algorithm is effective and superior to a particle swarm optimization in the literature.

Key words: permutation flowshop scheduling；NEH neighbourhood；Memetic algorithm

1 引 言

由于具有很强的实际工业背景, 流水车间调度问题 (PFSP)一直是最著名的生产调度问题之一, 受到了国内外众多学者的关注.

求解 PFSP 问题的方法可以分为 3 类: 精确算法、启发式算法和元启发式算法[1]. 由于最小化所有工件的总流水时间 (TFT) 的 PFSP 问题在机器数 $m \geqslant 3$ 时已被证明是 NP-完全问题, 因此, 用于求解 PFSP 问题的最优化方法较少, 主要包括混合整数规划方法[2]和分支定界算法[3]. 启发式算法主要包括构造式方法和改进式方法, 其中构造式方法主要是用于构造一个可行的解, 如 Johnson 算法[4]、NEH 算法[5]等; 改进式方法则是使用邻域搜索技术对构造式方法所得到的解进行迭代改进的方法[6]. 求解 PFSP 问题的超启发式算法主要包括遗传算法[7-9]、模拟退火算法[10-11]、蚁群算法[12]和粒子群 (PSO) 算法[13-15].

在以上算法中, 所使用的局部 (邻域) 搜索方法被证明对于整个算法的性能具有显著的影响, 而且超启发式算法被证明具有最好的性能. 因此, 设计有效的局部 (邻域) 搜索方法对于提高遗传算法、PSO 等算法的性能具有重要的意义. 在遗传算法等进化算法中加入局部 (邻域) 搜索方法后所得到的混合算法通常被称为 Memetic 算法. 近年来, 该算法已经得到了广泛的应用[16], 但是对于 PFSP 问题的相关应用还比较少[17].

本文针对 PFSP 问题, 提出了一个基于新型邻域结构的 Memetic 算法, 该邻域结构基于 NEH 算法的思想, 能够在算法的搜索过程中动态变化. 基于 PFSP 问题的标准 Benchmark 问题的测试结果表明了该算法的有效性.

收稿日期: 2012-05-03；**修回日期:** 2012-07-26.

基金项目: 国家自然科学基金项目(60804066, 60864004, 61034006).

作者简介: 徐建有(1971−), 男, 讲师, 博士生, 从事物联网、供应链网络的优化与可靠性的研究；顾树生(1939−), 男, 教授, 博士生导师, 从事先进控制技术等研究.

2 PFSP 问题描述

PFSP 问题可以简要描述如下: 在PFSP 问题中, 有一个待加工的工件集合 $J=\{1,2,\cdots,n\}$ 和一个机器集合 $M=\{1,2,\cdots,m\}$, 每个工件 $i\in J$ 必须在这 m 个机器上以相同的机器顺序进行加工, 即先从机器 1 开始加工, 然后是机器 2, 最后在机器 m 上完成加工. 每个工件 $i\in J$ 在机器 $j\in M$ 上的处理时间可以表示为 p_{ij}, 该值为固定值且非负. 假设工件在调度开始之前已经全部到达, 并且在生产过程中工件的加工不可中断. 在任何时刻, 每个工件只能在一个机器上进行加工, 并且每个机器一次最多只能加工一个工件. 此外, 每个工件只有在某一机器上完成了加工并且下一台相邻的机器空闲时, 该工件才能在相邻的下一台机器上开始加工. 目标是确定这 n 个工件在这 m 台机器上的排序, 以最优化一个特定的目标. 本文研究的优化目标是最小化 TFT, 因为该目标对于当前的动态生产环境和减少企业的在制品库存具有重要的意义.

令 $\pi=(\pi(1),\pi(2),\cdots,\pi(n))$ 表示一个工件的排序, 工件 $\pi(k)$ 在机器 j 上的完成时间可以利用下式进行计算:

$$C_{\pi(1),1}=p_{\pi(1),1}; \tag{1}$$

$$C_{\pi(1),j}=C_{\pi(1),j-1}+p_{\pi(1),j},\ j=2,3,\cdots,m; \tag{2}$$

$$C_{\pi(k),1}=C_{\pi(k-1),1}+p_{\pi(k),1},\ k=2,3,\cdots,n; \tag{3}$$

$$C_{\pi(k),j}=\max\{C_{\pi(k),j-1},\ C_{\pi(k-1),j}\}+p_{\pi(k),j},\quad k=2,3,\cdots,n,\ j=2,3,\cdots,m. \tag{4}$$

这样, 工件的总流水时间可以定义为

$$\mathrm{TFT}(\pi)=\sum_{k=1}^{n}C_{\pi(k),m}.$$

3 基于新邻域结构的 Memetic 算法

3.1 解的编码

对于 PFSP 问题, 解的一个很自然的编码方式为 n 个工件的排序, 即 $\pi=(\pi(1),\pi(2),\cdots,\pi(n))$, 其中 $\pi(k)$ 表示排在第 k 个位置的工件号.

3.2 种群初始化

在种群的初始化过程中, 为了保证初始种群能够具有较好的性能和分散性, 采用了两种方式来初始化种群: 1) 采用 Rajendran 等[18]提出的改进式方法来产生第一个解, 该解对于 TFT 目标具有很好的性能; 2) 随机产生种群 P 中的其他 N_P 个解.

3.3 种群更新

在种群初始化之后, 算法进入迭代进化过程. 在每一步迭代过程中, 算法从当前的种群 P 中随机选取两个解作为父代解, 通过两点交叉和变异算子得到一个新解. 这个过程需要重复进行 N_P 次, 以得到 N_P 个新解. 为了保证算法搜索的分散性同时兼顾质量, 与传统的进化算法不同, 在一个新解产生时,不使用该解更新父代解, 而是将其存储到一个临时的种群 P' 中; 然后, 针对包含 $2N_P$ 个解的种群集合 $P\bigcup P'$, 根据其目标函数值排序, 取前 N_P 个解作为下一迭代过程的初始种群. 该种群的更新过程如图 1 所示. 因为只有质量好的解才能进入下一次迭代过程, 所以该方法可以保证新种群解的质量; 同时, 由于解是由两个不同的种群得到的, 新种群的分散性也可以得到改善。

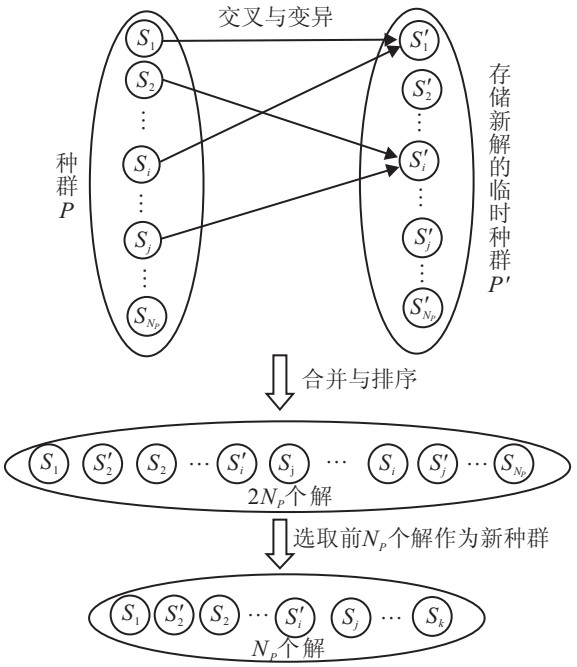

图 1 种群更新过程示意图

在新解的产生过程中, 所使用的交叉算子为两点交叉, 如图 2 所示. 该算子首先选定两个随机切点 r_a 和 $r_b(r_a<r_b)$; 其次将父代解 π_1 中位置在 $[1,r_a]$ 以及 $[r_b,n]$ 上的工件拷贝到新解中的相应位置; 然后, 从父代解 π_2 中删除已经安排在新解中的工件; 最后, 按照剩余工件在父代解 π_2 中的排序, 将它们拷贝到新解中的位置 $[r_a+1,r_b-1]$ 上. 该交叉算子经常被使用于求解调度问题的进化算法中.

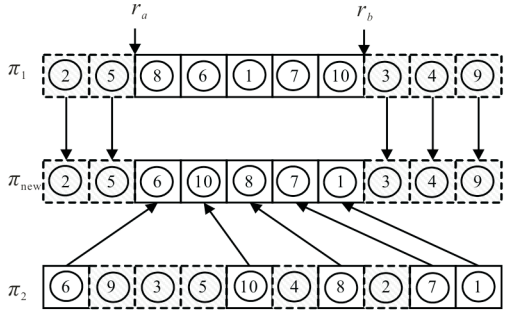

图 2 两点交叉算子示意图

变异算子使用 PFSP 算法中经常使用的 insertion 移动, 即首先从新解中随机删除一个工件, 然后再将

它随机插入其他位置上. 为了防止算法陷入局部最优, 经过实验, 采用2个连续的随机insertion移动作为变异算子. 这样可以保证解在变异后, 通过搜索较难回到其初始状态, 从而提高种群中解的分散性, 防止算法陷入局部最优.

3.4 基于NEH的邻域搜索

对于PFSP等组合优化问题, 局部搜索对于整个算法的性能有着显著的影响[19]. PFSP问题的解是工件的一个自然数序列, 因此针对该问题经常使用的邻域类型主要是insertion邻域和swap邻域. 其中insertion邻域基于insertion移动, 即从当前的解中删除一个工件, 并将它依次插入到解中的其他位置上. 通常的作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邻域最好解, 该邻域的计算复杂度为$O(n(n-1)) = O(n^2)$. 同样, swap邻域是基于swap移动, 即选定一个工件, 将其与其他工件进行交换, 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$O(n(n-1)/2) = O(n^2)$.

基于这种insertion和swap的移动, Tasgetiren等[13]在所提出的PSO算法中设计了一个变邻域搜索算法(VNS), 并取得了较好的效果. 这种VNS算法的缺点是计算复杂度高, 因此在这个PSO算法中, 作为局部搜索的VNS算法只是用于对当前种群中的最好解进行改进.

本文使用一种新的邻域结构, 该邻域结构基于insertion移动, 同时引入了NEH算法的思想. 对于一个解$\pi_0 = (\pi(1), \pi(2), \cdots, \pi(n))$, 基于该邻域结构的局部搜索过程如下:

Step 1: 令迭代次数$k = 1$, 设定当前最好解为$\pi_b = \pi_0$.

Step 2: 从解π_b中随机选择一个位置r, 再从这个位置开始, 连续删除d个工件. 如果$n-r+1 \geqslant d$, 则连续删除工件$\pi(r), \pi(r+1), \cdots, \pi(r+d-1)$; 否则, 删除工件$\pi(r), \cdots, \pi(n)$, 以及工件$\pi(1), \cdots, \pi(d-(n-r+1))$, 如图3所示. 为了以下描述的简便, 将被删除工件的顺序简记为$\pi(r_1), \pi(r_2), \cdots, \pi(r_d)$.

Step 3: 从$\pi(r_1)$开始, 将其插入到当前部分解中的最好位置上. 重复此插入过程, 直到所有被删除的工件全部重新插入到部分解中, 从而得到一个新的完整解π'.

Step 4: 如果目标函数值$\mathrm{TFT}(\pi') < \mathrm{TFT}(\pi_b)$, 则令$\pi_b = \pi'$.

Step 5: 令$k = k+1$, 如果$k > k_{\max}$(最大循环次数), 则停止; 否则, 转到Step 2.

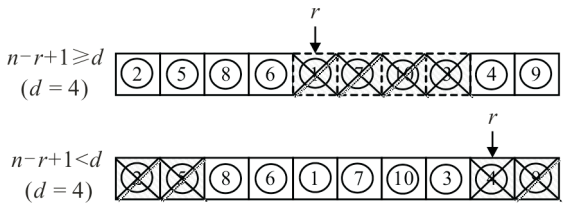

图 3　工件删除方法示意图

与传统的insertion移动相比, 该移动能够在更大的搜索空间进行搜索, 进而提高搜索到更好解的机会, 并提高算法跳出局部最优的能力. 当d取较大值时, 邻域规模变大; 反之, 则会变小. 因此, 为了使局部搜索能够在算法的初期具有良好的广域搜索能力, 在算法的后期具有良好的开发能力, 本文使用一种动态策略, 即d的取值随着算法搜索的进行而从大到小动态地变化. 该策略首先将算法的总运行时间或迭代次数平均分成d个区间(最后一个区间为算法即将结束的区间), 在第k个区间内连续删除工件的数目为$d-k+1$. 进行一次新型邻域搜索的复杂度为$O(k_{\max}dn)$.

此外, 在局部搜索算法中为了增强算法跳出局部最优的能力和广域搜索能力, 本文使用了循环迭代的方式, 并且在每次迭代中局部搜索算法会从一个随机的位置开始邻域搜索.

3.5 算法流程

基于以上描述, 可以给出所提出的Memetic算法的流程如下:

Step 1: 使用3.2节所描述的方法来初始化种群中的N_P个解, 即使用文献[12]中提出的改进式方法来产生第一个解, 其他的解则随机产生. 令当前最好解为种群中的最好解, 记为π_{best}, 令迭代次数$k = 0$, 当前算法的运行时间记为$t_{\mathrm{run}} = 0$, 当前的种群记为P_k.

Step 2: 令迭代次数$k = k+1$, 开始循环迭代.

Step 3: 按照3.3节中所描述的种群更新方法得到N_P个新解, 并将其与原来的种群P_k合并, 再按照其目标函数从小到大的顺序排列, 取前N_P个解作为新的种群.

Step 4: 如果当前的迭代次数$k\%5 = 0$(即每5次迭代), 则对种群P_k中前10个最好的解按照3.4节所描述的方法进行局部搜索.

Step 5: 如果当前种群中的最好解优于π_{best}, 则用其更新π_{best}.

Step 6: 如果算法达到了停止条件(如最大迭代次数或最大运行时间), 则停止; 否则, 转到Step 2继续进行迭代搜索.

如上所述, 该算法的创新点在于两个方面:

1) 在种群的更新过程中, 为了兼顾新种群的质量和分散性, 引入一个临时解种群, 用于保存通过交叉和变异算子所得到的新解. 通过将原种群和临时解种群合并和排序, 取质量较好的前 N_P 个解作为新种群.

2) 在所提出的Memetic算法中使用了基于NEH思想的动态邻域, 该邻域在算法搜索的过程中, 其邻域规模动态变小, 从而使算法在搜索的初期具有较好的广域搜索能力, 在算法的后期具有较好的局部搜索能力.

4 实验结果

为了验证算法的性能, 本文对算法进行了测试, 将其与其他文献中已有的方法进行了比较, 并且对算法中所采用的种群初始化方法以及新邻域结构的性能进行了实验测试和分析.

4.1 实验设置

本文选用Taillard等[20]提出的PFSP问题的标准Benchmark算例进行测试, 该测试集中包含9个规模, 其工件数从20个到100个, 机器数从5台到20台. 每个规模中包含10个算例, 共有90个算例.

为了与文献中已有的方法进行比较, 对于每个算例, 算法分别进行 $R = 10$ 次独立实验. 性能评价指标分别为最小相对偏差$\Delta_{\min}$, 最大相对偏差$\Delta_{\max}$, 以及平均相对偏差Δ_{avg}. 这些性能指标定义如下:

$$\Delta_{\min} = \min_{i=1,2,\cdots,R}\left\{\frac{(H_i - U_i)\times 100}{U_i}\right\}, \tag{5}$$

$$\Delta_{\text{avg}} = \sum_{i=1}^{R}\left(\frac{(H_i - U_i)\times 100}{U_i}\right)\Big/R. \tag{6}$$

其中: H_i 为算法所得到的解的目标函数值, U_i 为文献[21-22]所提出的两个算法(记为LR&RZ)所获得的最好解.

本文所提出的基于新邻域结构的memetic算法采用C++编程实现, 在CPU为Intel 2.8 GHz, 内存为4 GB, 操作系统为Windows XP的个人计算机上进行测试. 算法的停止准则取为最大运行时间 $T_{\max} = m \times n \times 0.1$ s. 在算法的运行过程中, 如果算法在求解的过程中得到了当前已知的最好解, 则算法提前终止. 算法的实验参数如下: 种群规模 $N_P = 100$, 连续删除的工件数 $d = 7$.

4.2 与其他算法的整体性能比较

除了LR&RZ方法外, 还与近几年所提出的性能较好的PSO算法[13]进行了比较, 该PSO算法记为PSO_{VNS}. 计算结果如表1所示, 表中 t_{avg} 表示平均的计算时间.

从表1中可以看出, 对于中小规模问题(工件数 $n \leqslant 50$ 且机器数 $m \leqslant 10$), PSO_{VNS} 算法表现出了较好的性能; 但是, 当机器数增大时($m = 20$), 本文所提出的Memetic算法显然具有更好的性能. 机器数 $m = 20$ 的问题被认为是所有Benchmark问题中最难求解的. 此外, 在大规模问题(工件数 $n \geqslant 100$)中, Memetic算法相对于PSO_{VNS}算法具有更好的性能. 在算法的平均性能和所获得的最好解的质量上, 本文所提出的Memetic算法也明显优于以往文献中的PSO_{VNS}算法, 并且Memetic算法所需要的运行时间也明显减少. 因此, 基于新型邻域结构的局部搜索的性能要明显优于PSO_{VNS}算法中所使用的VNS算法.

表 1 与其他算法的性能比较结果

问题	PSO_{VNS} 算法			Memetic 算法		
	Δ_{avg}	$\Delta_{\min}$	t_{avg}	Δ_{avg}	$\Delta_{\min}$	t_{avg}
20×5	**−0.12**	**−0.17**	3.18	−0.08	**−0.17**	5.70
20×10	**0.04**	**−0.04**	7.21	**0.04**	**−0.04**	9.88
20×20	2.96	2.76	11.93	**−0.01**	**−0.07**	17.78
50×5	**−0.38**	**−0.60**	41.71	0.01	−0.37	25.03
50×10	**−0.41**	**−0.82**	74.49	−0.32	−0.75	50.03
50×20	1.40	0.86	143.32	**−0.26**	**−0.68**	100.04
100×5	**−0.40**	**−0.57**	222.28	−0.11	−0.40	50.07
100×10	−0.28	−0.69	407.88	**−0.46**	**−0.85**	100.09
100×20	0.29	−0.10	824.41	**−0.61**	**−1.08**	200.16
平均	**0.34**	**0.07**	**192.93**	**−0.20**	**−0.49**	**62.09**

4.3 基于NEH的邻域与其他传统邻域的比较

为了进一步说明所使用的新邻域结构的有效性, 在实验中还将其与传统的insertion邻域和swap邻域进行了性能对比, 即在本文的Memetic算法的局部搜索中分别只使用传统的insertion邻域和swap邻域, 计算结果如表2所示.

表 2 与传统邻域结构的性能比较

问题	Insertion 邻域		Swap 邻域		基于NEH的新邻域	
	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$
20×5	0.26	0.04	0.13	−0.06	−0.08	**−0.17**
20×10	0.25	0.04	0.28	0.01	0.04	**−0.04**
20×20	0.23	–0.01	0.19	0.00	**−0.01**	**−0.07**
50×5	0.73	0.31	0.18	−0.17	0.01	-0.37
50×10	0.69	0.19	0.41	−0.20	-0.32	-0.75
50×20	0.82	0.25	0.66	0.07	**−0.26**	**−0.68**
100×5	0.55	0.27	0.02	−0.25	−0.11	−0.40
100×10	0.66	0.27	0.28	−0.09	**−0.46**	**−0.85**
100×20	0.67	0.22	0.64	0.18	**−0.61**	**−1.08**
平均	**0.54**	**0.18**	**0.31**	**−0.06**	**−0.20**	**−0.49**

从表2中可看出, swap邻域的性能在平均性能和最好解的平均质量上都优于insertion邻域. 相对于传统的insertion邻域和swap邻域, 基于NEH思想的新邻域表现出了更好的性能. 究其原因, 传统邻域结构的搜索步伐较小, 导致算法很容易陷入局部最优, 同时它们的邻域规模也相对较小; 而基于NEH思想的新邻域结构则克服了以上缺点, 具有更好的搜索性能.

4.4 基于NEH的不同邻域规模性能比较

为了更好地分析新邻域的性能, 本文对其不同的

规模进行了测试, 即连续删除的工件数分别取为{3, 5, 7}, 测试结果如表3所示.

表 3　不同邻域规模的性能比较

问题	$d=3$		$d=5$		$d=7$	
	$\Delta_{\rm avg}$	$\Delta_{\rm min}$	$\Delta_{\rm avg}$	$\Delta_{\rm min}$	$\Delta_{\rm avg}$	$\Delta_{\rm min}$
20×5	0.00	−0.14	−0.04	**−0.17**	−0.08	**−0.17**
20×10	0.07	0.00	0.06	−0.02	**0.04**	**−0.04**
20×20	0.04	−0.03	0.01	**−0.07**	**−0.01**	**−0.07**
50×5	0.12	−0.26	0.03	−0.34	**0.01**	**−0.37**
50×10	−0.18	−0.61	**−0.33**	−0.71	−0.32	**−0.75**
50×20	−0.10	−0.58	−0.21	**−0.73**	**−0.26**	−0.68
100×5	0.01	−0.23	−0.08	−0.34	**−0.11**	**−0.40**
100×10	−0.24	−0.61	−0.41	−0.82	**−0.46**	**−0.85**
100×20	−0.39	−0.81	**−0.62**	−1.01	−0.61	**−1.08**
平均	−0.07	−0.36	**−0.18**	**−0.47**	**−0.20**	**−0.49**

从表3中可以看出, 随着邻域规模的增大, 算法的性能逐渐变好, 这是因为邻域规模越大, 局部搜索的搜索空间越大, 得到更好解的机率更大, 同时算法跳出局部最优的能力也越强. 虽然这需要耗费更多的计算资源, 但是所获得的比较结果表明, $d=7$ 是一个比较理想的取值. 在实验中也测试过 d 取更大的值, 但是结果表明, 当邻域规模过大时, 算法的搜索性能反而会下降, 这是因为邻域规模过大时局部搜索所需要的时间会大大增加, 从而导致算法的广域搜索不足, 进而导致算法整体搜索性能的下降.

4.5　与当前最好解之间的性能比较

在以上的实验中, 将本文提出的Memetic算法与其他文献中已有的算法进行了性能比较, 比较结果表明了算法的有效性. 然后将该算法所取得的最好结果与当前已知的最好解进行比较, 进一步证明了算法的有效性. 当前已知的最好结果为Taillard在网站上公布的结果[23], 这些解均是从多个不同智能优化算法所得到的最好解的集合中选取的最好解.

比较结果如表4所示, 从结果中可以看出, 在平均性能上, 本文所提出的Memetic算法与当前已知的最好结果之间的偏差在0.2%左右, 已经非常接近于当前最好解. 此外, 在实验中对于10个算例, 本文提出的Memetic算法得到了更好的解, 这10个算例以及它们对应的新目标值如表5所示.

表 4　与当前最好结果的性能比较

问题	当前最好结果	Memetic 算法
20×5	0.00	0.01
20×10	0.00	0.00
20×20	0.00	0.00
50×5	0.00	0.49
50×10	0.00	0.58
50×20	0.00	0.33
100×5	0.00	0.31
100×10	0.00	0.11
100×20	0.00	**−0.03**
平均	0.00	0.20

表 5　算例的新目标值

算 例	当前最好目标值	Memetic 算法	改进量/%
Taillard-60	124 866	**124 660**	**−0.16**
Taillard-69	249 785	**249 616**	**−0.07**
Taillard-75	287 626	**286 538**	**−0.38**
Taillard-78	293 572	**293 524**	**−0.02**
Taillard-80	294 705	**294 516**	**−0.06**
Taillard-81	371 121	**370 330**	**−0.21**
Taillard-82	376 476	**375 919**	**−0.15**
Taillard-84	377 221	**375 626**	**−0.42**
Taillard-89	378 231	**378 174**	**−0.02**
Taillard-90	383 840	**382 099**	**−0.45**

5　结　论

针对PFSP问题, 本文提出了一个基于新型邻域结构的Memetic算法, 该算法的主要贡献是采用了基于NEH思想的新型邻域结构, 并且在局部搜索的过程中使该邻域的结构动态变化, 以保证良好的搜索性能. 此外, 在种群的更新过程中, 从两个父代解得到一个新解时, 不是用其马上更新父代解, 而是将其存储到一个临时种群中, 然后从两个种群的并集中选择质量好的解形成新的种群, 以兼顾种群的质量和分散性. 使用PFSP问题的标准Benchmark算例对所提出的算法进行了实验测试, 并与其他文献中的算法进行了比较, 结果显示本文算法在解的质量和求解时间上明显优于以往文献中提出的2个启发式方法和1个PSO算法, 充分表明了本文算法的有效性. 通过与传统邻域结构的性能比较也显示了新型邻域结构具有更好的搜索寻优能力. 与当前最好结果的比较表明, 本文算法与这些结果之间的平均偏差在0.2%左右. 此外, 对于10个算例本文算法还得到了更好的结果, 充分表明了算法的有效性.

下一步的研究重点是对于种群的分散性进行更好的控制, 提出更好的种群更新策略, 并将该算法在实际工程与决策问题中进行应用验证.

参考文献(References)

[1] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation fowshop heuristics[J]. European J of Operational Research, 2005, 165(2): 479-494.

[2] Stafford E F. On the development of a mixed integer linear programming model for the flowshop sequencing problem[J]. J of the Operational Research Society, 1988, 39(12): 1163-1174.

[3] Lomnicki Z A. A branch and bound algorithm for the exact solution of the three machine scheduling problem[J]. Operational Research Quarterly, 1965, 16(1): 89-100.

[4] Campbell H G, Dudek R A, Smith M L. A heuristic algorithm for the n job, m machine sequencing problem[J]. Management Science, 1970, 16(10): 630-637.

[5] Nawaz M, Enscore E E, Ham I. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem[J]. OMEGA 1983, 11(1): 91-95.

[6] Suliman S. A two-phase heuristic approach to the permutation flow-shop scheduling problem[J]. Int J of Production Economics, 2000, 64(1/2/3): 143-152.

[7] Reeves C R. A Genetic algorithm for flowshop sequencing[J]. Computers & Operations Research, 1995, 22(1): 5-13.

[8] Ruiz R, Maroto C, Alcaraz J. Two new robust genetic algorithms for the flowshop scheduling problem[J]. OMEGA, 2006, 34(5): 461-476.

[9] Zobolas G I, Tarantilis C D, Ioannou G. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm[J]. Computers & Operations Research, 2009, 36(4): 1249-1267.

[10] Osman I H, Potts C N. Simulated annealing for permutation flow-shop scheduling[J]. OMEGA, 1989, 17(6): 551-557.

[11] Ishibuchi H, Misaki S, Tanaka H. Modified simulated annealing algorithms for the flow shop sequencing problem[J]. European J of Operational Research, 1995, 81(2): 388-398.

[12] Rajendran C, Ziegler H. Ant colony algorithms for permutation flowshop scheduling to minimize makespan/ total flowtime of jobs[J]. European J of Operational Research, 2004, 156(2): 426-438.

[13] Tasgetiren M F, Liang Y C, Sevkli M, et al. A particle swarm optimization algorithm for makespan and total fowtime minimization in the permutation flowshop sequencing problem[J]. European J of Operational Research, 2007, 177(3): 1930-1947.

[14] Jarboui B, Ibrahim S, Siarry P, et al. A combinatorial particle swarm optimisation for solving permutation flowshop problems[J]. Computers & Industrial Engineering, 2008, 54(3): 526-538.

[15] Liao C J, Tseng C T, Luarn P. A discrete version of particle swarm optimization for flowshop scheduling problems[J]. Computers & Operations Research, 2007, 34(10): 1899-1909.

[16] Neri F, Cotta C. Memetic algorithms and memetic computing optimization: A literature review[J]. Swarm and Evolutionary Computation, 2012, 2(1): 1-14.

[17] Li X, Wang J, Zhou J, et al. An effective GSA based memetic algorithm for permutation flow shop scheduling[C]. Proc of 2010 IEEE Congress on Evolutionary Computation. New York: IEEE Press, 2010: 1-6.

[18] Rajendran C, Ziegler H. A heuristic for scheduling to minimize the sum of weighted flowtime of jobs in a flowshop with sequence-dependent setup times of jobs[J]. Computers & Industrial Engineering, 1997, 33(1/2): 281-284.

[19] Ishibuchi H, Yoshida T, Murata T. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling[J]. IEEE Trans on Evolutionary Computation, 2003, 7(2): 204-223.

[20] Taillard E. Benchmarks for basic scheduling problems[J]. European J of Operational Research, 1993, 64(2): 278-285.

[21] Liu J Y, Reeves C R. Constructive and composite heuristic solutions to the P//$\sum C_i$ scheduling problem[J]. European J of Operational Research, 2001, 132(2): 439-542.

[22] Rajendran C, Ziegler H. Ant colony algorithms for permutation flowshop scheduling to minimize makespan/ total flowtime of jobs[J]. European J of Operational Research, 2004, 155(2): 426-438.

[23] Taillard E. Summary of best known lower and upper bounds of Taillard's instances[EB/OL]. (2012-07-03) [2005-04-13]. http://mistic.heig-vd.ch/taillard/problemes. dir/ordonnancement. dir/ordonn-ancement.html.

(上接第1780页)

[14] 曾庆山, 曹广益. 分数阶线性系统的能观性研究[J]. 系统工程与电子技术, 2004, 26(11): 1647-1650.
(Zeng Q S, Cao G Y. Research on observability of the fractional-order linear systems[J]. Systems Engineering and Electronics, 2004, 26(11): 1647-1650.)

[15] Canudas W C, Olsson H, Astrom K J, et al. A new model for control of system with friction[J]. IEEE Trans on Automatic Control, 1995, 40(3): 419-425.