求解流水车间调度问题的混合粒子群算法

田 野[1,2],刘大有[1,2]
(1. 吉林大学计算机科学与技术学院,吉林长春 130012;2. 吉林大学符号计算与知识工程教育部重点实验室,吉林长春 130012)

摘 要: 本文提出了一种混合的元启发式方法 HDCPSO 用于求解置换流水车间调度问题中的最小化完成时间.该算法将粒子群算法和迭代贪心算法(Iterative Greedy,IG)相结合,利用 IG 算法中的作业毁坏(Destruction)和构造(Construction)操作来对粒子进行变异,降低群体发生早熟的可能.引入了个体徘徊概念,用来控制个体变异.此外,通过基于插入的邻域搜索来提高个体的局部搜索能力.最后,提出了群体的重新初始化机制来进一步避免早熟收敛的发生.本文算法在不同规模的问题实例上与其他几个具有代表性的算法进行了比较,实验结果表明,无论是在求解质量还是稳定性方面都优于其他算法.

关键词: 粒子群算法; 车间调度; 迭代贪心算法; 个体徘徊; 重新初始化
中图分类号: TP18 **文献标识码:** A **文章编号:** 0372-2112 (2011) 05-1087-07

A Hybrid Particle Swarm Optimization Method for Flow Shop Scheduling Problem

TIAN Ye[1,2], LIU Da-you[1,2]
(1. *College of Computer Science and Technology, Jilin University, Changchun, Jilin* 130012, *China*;
2. *Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun, Jilin* 130012, *China*)

Abstract: A hybrid metaheuristic for the minimization of the makespan in permutation flow shop scheduling problem (FSSP) is proposed, which combines Particle Swarm Optimization (PSO) and Iterative Greedy (IG), and takes advantage of Destruction and Construction (DC) of IG to prevent the swarm from premature convergence. Individual hovering is defined to control when the particle mutate. Besides, an insert (shift) neighborhood search is introduced to improve the particle's searching ability. Finally, we present an efficient population re-initialization scheme to avoid premature convergence further. The proposed algorithm is tested on different scale benchmarks and compared with the other representative algorithms. The result shows that HDCPSO is better than other algorithms in not only the solution quality but also the stability.

Key words: particle swarm optimization; flow shop scheduling; iterative greedy; individual hovering; re-initialization scheme

1 引言

流水车间调度问题通常可以描述为 n 个作业要在 m 台机器上加工,每个作业有 m 道工序,每道工序都要在不同的机器上加工,n 个作业在 m 台机器上的加工顺序相同,问题的目标是求 n 个作业在每台机器上最优的加工顺序,使得特定的性能指标得到满足.置换流水车间调度(Permutation Flow Shop Scheduling Problem,PFSP)是对典型的流水车间调度问题的进一步约束,即约定每台机器上所有作业的加工顺序都相同,尽管 PFSP 工艺约束比较简单,但是已经被证明是 NP-Complete 问题[1].本文主要研究 PFSP 中的最大完成时间问题,最大完成时间是生产调度中最常用的性能度量指标之一,最大完成时间越短,则说明产品总的生产周期越短,生产能力越大,即最小化最大完成时间意味着最大化生产能力.因此,对其进行优化调度可有效地提高企业生产效益和资源利用率.对于 n 个作业,m 台机器的 PFSP,作业集合 $J=\{J_j \mid j=1,2,\cdots,n\}$,$t_{i,j}$表示作业 j 在机器 i 上的加工时间($j=1,2,\cdots,n$; $i=1,2,\cdots,m$),$C(i,J_j)$ 表示第 j 个作业在第 i 台机器上的完成时间,那么最大完成时间 $C_{\max}$可以通过下面的数学公式来表述:

$$C(1,J_1)=t_{1,J_1} \tag{1}$$

$$C(1,J_j)=C(1,J_{j-1})+t_{1,J_1}(j=2,3,\cdots,n) \tag{2}$$

收稿日期:2010-03-30;修回日期:2010-06-07
基金项目:国家自然科学基金(No.60603030, No.60773099, No.60873149, No.60973088);国家 863 高技术研究发展计划(No.2006AA10Z245, No.2006AA10A309);欧盟项目(Bridging the Gap, No.155776-EM-1-2009-1-IT-ERAMUNDUS-ECW-L12)

$$C(k,J_1)=C(k-1,J_1)+t_{k,J_1}(k=2,3,\cdots,m) \quad (3)$$

$$C(k,J_j)=\max\{C(k-1,J_j),C(k,J_{j-1})\}+t_{k,J_j} \quad (4)$$

$$C_{\max}=C(J_n,m) \quad (5)$$

粒子群算法(Particles Swarm Optimization,PSO)是由Kennedy和Eberhart博士于1995年提出的一种基于群体的进化算法[2],该算法主要模拟鸟群的行为.粒子群算法通用性强,易于实现,因此被广泛应用于不同的领域.相对于其他的一些算法,如遗传算法(Genetic Algorithm,GA)、模拟退火算法(Simulated Annealing,SA)等,应用PSO求解车间调度问题的研究还相对较少.文献[3]最先对于PFSP问题提出了一种基于可变邻域搜索(Variable Neighborhood Search,VNS)的混合粒子群算法PSOVNS,该算法采用基于随机键的规则来构造粒子的位置到作业排序之间的映射关系.该算法后来被用于求解最小化最大完成和总的流经时间[4].Liao等人[5]扩展传统的二元离散粒子群算法用于求解流水车间调度问题,并且和遗传算法进行了对比,实验证明要优于遗传算法.Lian等人[6]提出了最小化最大完成时间的PFSP的一种基于PSO思想的进化算法,他们直接利用GA的交叉和变异操作作为PSO算法中粒子速度和位置的更新操作.高亮等人[7]提出新的基于粒子群的调度算法,该算法同样采用遗传操作,主要利用记忆库来实现信息共享机制,降低了算法局部收敛的概率.文献[8]中提出的粒子群算法的思想和文献[6]一样,但是通过对部分粒子的位置和速度进行变异来降低早熟收敛现象发生的可能.文献[9]提出了一种改进的离散粒子群算法,并将其用于车间调度问题.

本文提出了一种混合的元启发式方法HDCPSO用于求解置换流水车间调度问题中的最小化最大完成时间问题.该算法结合粒子群算法以及迭代贪心算法(Iterative Greedy,IG),利用IG算法中的作业毁坏(Destruction)和构造(Construction)操作(本文简称DC操作)来对粒子(为了方便,本文不区分粒子和个体的概念)进行变异,降低群体发生早熟的可能.并且引入了个体徘徊概念,用来控制个体变异.此外,通过基于插入的邻域搜索来强化个体的局部搜索能力.最后,本文还提出了群体的重新初始化机制来进一步避免群体的早熟收敛的发生.

2 IG算法

Ruiz等人[10]提出了一种简单的迭代贪心算法(Iterated Greedy Algorithm,IG)用于流水车间调度问题,该算法利用NEH的基本原理,通过对作业排列的DC操作来获得更好的解.给定一个作业排列π,IG算法首先对作业排列进行毁坏处理,即随机地、无替换地从作业排列π中删除若干作业,然后将删除的作业依次重新插回到可能的最好的位置,这样就重新获得了一个完整的作业排列,然后与毁坏前的作业排列进行比较,将满足条件的作业排列作为当前解,进入下一轮迭代.

3 HDCPSO算法

在本节中,我们将粒子群算法与IG算法相结合,提出一种混合的粒子群算法HDCPSO.HDCPSO算法的基本框架如图1所示.图1描述了算法的基本组成部分,我们将在下面的几个小节中具体介绍这些组成部分.

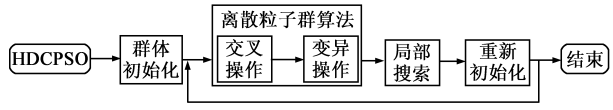

图1 HDCPSO算法基本框架

3.1 群体初始化

NEH作为一种求解PFSP问题的启发式构造型算法[11],经常被用于群体的初始化,以实现快速收敛.本文中通过NEH来构造一个粒子,剩下的粒子随机生成.

3.2 速度和位置更新

传统的粒子群速度和位置更新策略无法直接应用于离散问题,但是可以通过交叉操作来实现相同的目的,本文算法中粒子的位置和速度直接表示成作业排列,速度和位置的更新公式如下:

$$V_i(t+1)=V_i(t)\otimes P_{ibest}\otimes P_{gbest} \quad (6)$$

$$X_i(t+1)=V_i(t+1)\otimes X_i(t) \quad (7)$$

其中t表示迭代次数,$\otimes$表示交叉操作符.从上面公式可以看出,粒子的行为依然受到粒子速度、当前个体最优和当前群体最优(以下简称为个体最优和群体最优)的影响,并通过信息的共享向最优解方向前进.在更新速度时使用群体最优容易导致早熟收敛,因此我们先选择适应度最好的一部分个体(本文选取5个),然后从中随机选择一个作为更新时使用的群体最优.

3.3 交叉操作

交叉操作通过交换父代个体的信息,以期望获得更好的子代个体.Pan等人[12]提出一种交叉方式,two-cut PTL crossover(简称PTL).该交叉方式和文献[8]中的C1交叉方式类似,不同的是将作业子排列分别复制到子代个体的前面或后面.这种交叉方式的好处是即使两个父代个体的作业排列是一样的,交叉后也可以得到两个不同的子代个体.受此启发,本文提出了一种随机放置的交叉方式,称为two-cut Random PTL crossover(简称RPTL),即两个切点之间的作业子排列可以放置在子代中的任何位置,只要保证子排列在两个子代个体中的位置对称即可.两种交叉方式的区别见表1.

RPTL具备PTL的特点,并且避免了PTL由于交叉

程度过大而使得子代个体远离更好的区域,降低收敛速度.当随机选择的位置恰好是子代中的第一个位置或者是最后一个位置,那么 RPTL 交叉就转化成 PTL 交叉,因此可以认为 RPTL 交叉是 PTL 交叉的一般形式.

在实验中,我们测试了三种交叉方式的影响,即 C1 交叉、C1-1 交叉(RPTL 和 C1 交替执行)和 C1-2 交叉(先执行 C1 交叉,当两个排列一致时采用 RPTL 交叉),发现 C1-2 交叉在求解速度上接近 C1 交叉,并且求解质量优于其他两种交叉方式,因此我们认为 RPTL 交叉策略可以弥补 C1 的不足,将两个策略结合起来是行之有效的,并且在后面实验中采用 C1-2 交叉方式.

表 1　PTL 和 RPTL 交叉的对比

Two-cut PTL crossover								Two-cut RPTL crossover							
Parent1	3	5	**1**	**7**	**2**	6	4	Parent1	3	5	**1**	**7**	**2**	6	4
Parent2	3	5	**1**	**7**	**2**	6	4	Parent2	3	5	**1**	**7**	**2**	6	4
Child1	**1**	**7**	**2**	3	5	6	4	Child1	3	**1**	**7**	**2**	5	6	4
Child2	3	5	6	4	**1**	**7**	**2**	Child2	3	5	6	**1**	**7**	**2**	4

3.4　变异操作

变异操作是通过对当前个体的扰动来产生一个新的个体,从而避免早熟收敛.如果一个粒子的个体最优连续若干代都没有发生变化,即无法移动到更好的位置而发生停滞,我们称之为个体徘徊.

定义 1　个体徘徊:对于一个粒子,如果连续 p 代,其个体最优都没有发生变化,则称为个体徘徊.

我们将前面介绍的 IG 算法与粒子群算法结合起来,通过 IG 算法中的 DC 操作对粒子进行变异操作.IG 算法与迭代的局部搜索非常相似,只是 IG 是一种迭代的构造启发式算法,利用 DC 操作寻求新的个体最优解,减少个体陷入局部极值的概率.

由于相对于简单的变异操作,DC 操作需要更长的计算时间,因此我们只针对发生个体徘徊现象的粒子的个体最优进行变异.对每个个体,利用一个计数器来记录个体最优停滞的代数,在初始化阶段,每个粒子的计数器都为 0.个体每停滞一代,则计数器加 1,否则重新计数.如果达到阈值 p,则对个体最优进行变异操作,然后重新开始计数.

3.5　局部搜索

大量的研究发现,粒子群算法的局部搜索能力较弱,尤其在进化后期,进化变得缓慢,甚至停止进化发生早熟收敛.文献[13]采用插入邻域搜索,对每个个体的邻域进行搜索,提高了算法的收敛速度和精度.文献[14]中采用了一种基于引用的插入模式(Referenced insertion scheme)来进行局部搜索,利用一个引用作业排列(通过 NEH 获得)来引导局部搜索.本文中采用这种方式,不同的是我们将适应度最差的粒子对应的作业排列作为引用作业排列,这样可以使得每次局部搜索时所采用的引用作业排列都有可能不同.由于局部搜索会导致计算时间的增加,因此,我们只对当前全局最优个体采用插入邻域搜索.

3.6　重新初始化

随着群体的不断进化,群体多样性不断降低,使得群体容易停滞在局部最优,本文采取重新初始化机制来克服该问题.当群体最优的停滞代数达到指定的阈值时,执行重新初始化操作,增强群体多样性.重新初始化过程分为删减阶段和增添阶段.

3.6.1　删减阶段

为了不影响算法的收敛速度,不需要对整个群体全部进行删减,而是从个体适应度和群体多样性两个方面来考虑,这样即保证最好的个体不被删除,同时也删除了一部分相似度比较接近的个体,以维护整个群体的多样性,避免陷入局部极值.第一步:对整个群体,根据个体的适应度优劣,我们先保留适应度最好的前 20% 的个体;第二步:删除适应度最差的 30%;第三步:删掉多样性相对较差的 20%.在第三步操作中,首先从保留下来的适应度最好的那 20% 中随机选择一个作为基准个体,然后从前两步中未被处理的那 50% 个体中随机选择一些(本文中取 2)和基准个体进行相似度比较,删掉和基准个体之间相似度较大的那个个体,重复执行该过程,直至总群体数中的 20% 被删掉.

定义 2　个体相似度:对于两个个体 a 和 b,其作业排列分别为 π^a 和 π^b,那么这两个个体的相似度定义为两个个体作业排列间的 Hamming 距离,记作 $\mathrm{Ham}(\pi^a,\pi^b)$.

$$\mathrm{Ham}(\pi^a,\pi^b)=\sum_{j=1\cdots n}\mathrm{sign}(|\pi_j^a-\pi_j^b|) \tag{8}$$

其中 sign 是一个符号函数.

3.6.2　增添阶段

在增添阶段,我们采用分散搜索(Scatter Search,SS)中的群体多样性生成策略[15]和随机生成两种方式来重新初始化被删减的粒子.通过分散搜索中的群体多样性生成策略产生 20% 个体(如果生成个体数不足 20%,则在前 20% 适应度较高的个体中随机选择一个,然后执行 Shift 插入操作来补足),随机产生 30% 的个体.

3.7　HDCPSO 算法描述

基于前面几部分的介绍,本文提出的 HDCPSO 算法的伪代码描述如下:

```
Procedure HDCPSO
Input:测试实例;群体大小 NP 以及其他参数;
Output:最佳作业排列和最小完成时间;
```

```
Begin
    t←0;// 第 0 代
    初始化群体 Pop(t);
    // 更新个体最优和群体最优
    For i = 1 to NP Do
    Begin
        P_ibest←X_i(t);
        Count(P_ibest)←0;
    End;
    P_gbest←BestFit(Pop(t));
    While 不满足终止条件 Do
    Begin
    For i = 1  to NP Do
        Begin
            // 更新个体的速度和位置
            V_i(t+1) = V_i(t) ⊗ P_ibest ⊗ P_gbest
            X_i(t+1) = V_i(t+1) ⊗ X_i(t)
            // 更新个体最优和群体最优
            If C_max(X_i(t+1)) < C_max(P_ibest) Then
                P_ibest←X_i(t+1);
            Else
    Count(P_ibeste)←Count(P_ibeste) + 1;
            If Count(P_ibest) = p Then
                // 执行 IG 中的 DC 操作
                P_ibest←DC(P_ibest);
            If C_max(P_ibest) < C_max(P_gbest) Then
                P_gbest←P_ibest;
        End;
        // 执行局部搜索
        Referenced_Local_Search(P_gbest);
        If Count(P_gbest) = q Then
            ReInit (Pop(t));  // 重新初始化
        t←t + 1;
    End;
    输出群体最优 P_gbest;
End.
```

4 实验

在本节中,首先分析了不同的参数设置对算法性能的影响,由于本文中主要使用两个参数,即 DC 变异时需要毁坏的作业个数和个体徘徊代数,因此我们主要通过实验对个体徘徊代数进行分析.然后,在不同规模的 Taillard 数据集上对本文算法进行了测试,并和 NEH 以及 NPSO 算法[8]做了对比.最后和 Kuo 等人最新提出的混合粒子群算法 HPSO[16]进行了比较.

4.1 参数设置

在本文算法中,关键的参数主要有两个,一是当个体最优执行 DC 操作时毁坏作业的个数;二是个体徘徊代数,下面我们分别来讨论这两个参数.

4.1.1 毁坏作业个数

在文献[10]中,Ruiz 等人通过大量的实验来测试不同毁坏作业个数对算法的影响,并最终得出毁坏个数为 4 时相对性能最好.因此,本文直接采用文献[10]中的结果.

4.1.2 个体徘徊代数

本小节我们来分析一下个体徘徊代数对算法的影响.为了测试徘徊代数的影响,我们将个体徘徊代数 P 分别设置为 5,10,20,30,40,50,群体最优停滞代数设置为 50,毁坏作业个数为 4,群体规模设置为 60,最大迭代次数为 500 代,问题实例采用 TA021、TA051 和 TA081.对于每个测试,算法独立执行 20 次.每次测试所得到的最优解与已知最优解的平均相对偏离度(Average Relative Percentage Deviation,ARPD)定义如下:

$$ARPD = \sum_{i=1}^{L}\left(\frac{(Sol_i - BKS)\times 100}{BKS}\right)\Big/L \qquad (9)$$

其中 L 表示算法执行的次数,Sol_i 表示算法在第 i 次执行时获得的最优解,BKS 表示已知的问题最优解,因此 ARPD 能反映出算法得到的最优解与真实最优解之间的相对偏离程度.徘徊代数对 ARPD 的影响见表 2.

表 2 徘徊代数取不同的值的影响

Problem	Size	$P=5$	$P=10$	$P=20$	$P=30$	$P=40$	$P=50$
TA021	20×20	0.471	0.366	0.298	0.385	0.644	0.435
TA051	50×20	2.117	1.984	1.813	1.907	2.083	2.134
TA081	100×20	2.607	2.477	2.249	2.795	2.918	2.757

从表 2 中可以看出,对于所有问题,徘徊代数为 20 时都得到了较好的结果,因此本文算法中个体徘徊代数设置为 20.

4.2 实验结果

4.2.1 实验对比 1

首先,我们将 HDCPSO 算法与 NEH 以及 NPSO 算法进行对比,NPSO 算法采用 C1 交叉,变异方式为移位变异(M3).HDCPSO 算法采用 C1-2 交叉,毁坏作业个数为 4,个体徘徊代数为 20,群体最优停滞代数为 50,群体大小为 60,最大迭代次数为 500.初始化时均采用由 NEH 生成一个个体,而剩余的随机生成.测试问题的大小由 20×5 到 100×20,每个测试实例执行 10 次.

三个算法均采用 C # 语言实现,机器配置为 windows XP 系统,处理器为 Pentium 4,2.8GHz,内存大小为 1G.表 3~5 列出了作业个数为 20、50 和 100 时算法的对比结果.

从表 3~5 中,我们可以看出,对于 9 个不同规模的测试问题,本文提出的 HDCPSO 算法,获得的结果(最优解、最差解和平均值)均优于 NEH 和 NPSO 算法.

表 3　NEH、NPSO 和 HDCPSO 对于 **20** 个作业的测试问题的对比结果

Problem	BKS	NEH	NPSO	HDCPSO
			Min/Max/Avg	Min/Max/Avg
TA001	1278	1286	1278/1286/1280.5	1278/1278/1278
TA005	1235	1305	1243/1250/1246.8	1235/1235/1235
TA010	1108	1151	1117/1131/1127.3	1108/1108/1108
TA011	1582	1680	1591/1627/1601.6	1582/1593/1585.6
TA015	1419	1502	1453/1466/1462.4	1419/1431/1423.1
TA020	1591	1653	1608/1636/1626.3	1591/1608/1598.4
TA021	2297	2410	2317/2333/2325.8	2297/2310/2303.5
TA025	2291	2397	2321/2379/2335.5	2291/2304/2295.3
TA030	2178	2277	2199/2240/2206.6	2179/2190/2182

表 4　NEH、NPSO 和 HDCPSO 对于 **50** 个作业的测试问题的对比结果

Problem	BKS	NEH	NPSO	HDCPSO
			Min/Max/Avg	Min/Max/Avg
TA031	2724	2733	2729/2729/2729	2724/2724/2724
TA035	2863	2868	2864/2864/2864	2863/2864/2863.3
TA040	2782	2790	2784/2784/2784	2782/2782/2782
TA041	2991	3135	3094/3108/3101.4	3034/3047/3036.1
TA045	2976	3160	3049/3134/3080.8	3008/3024/3016.9
TA050	3065	3257	3190/3213/3201	3108/3140/3116.5
TA051	3850	4082	3989/4020/3998.6	3898/3972/3924.8
TA055	3610	3835	3716/3806/3772.6	3658/3717/3689.5
TA060	3756	4079	3913/3994/3962.8	3806/3844/3816.9

表 5　NEH、NPSO 和 HDCPSO 对于 **100** 个作业的测试问题的对比结果

Problem	BKS	NEH	NPSO	HDCPSO
			Min/Max/Avg	Min/Max/Avg
TA061	5493	5519	5493/5519/5499.3	5493/5493/5493
TA065	5250	5266	5252/5266/5256.4	5250/5252/5251.2
TA070	5322	5341	5335/5341/5340.3	5322/5324/5322.4
TA071	5770	5846	5803/5846/5814.8	5771/5799/5787.4
TA075	5467	5679	5539/5559/5553.6	5491/5520/5504.2
TA080	5845	5918	5903/5903/5903	5848/5903/5881
TA081	6202	6541	6451/6530/6496.5	6309/6369/6352
TA085	6314	6695	6583/6652/6616.1	6419/6486/6451.4
TA090	6434	6677	6600/6657/6629.4	6539/6570/6551.2

对于小规模的问题(作业数为 20),除了在 TA030 问题实例上,HDCPSO 算法都能够获得和已知最优解相同的结果,而 NPSO 只在实例 TA001 上获得了最优解.对于中等及大规模问题上,当机器数量比较少时,HDCPSO 算法对于所有的测试问题实例都获得了最有解,而 NPSO 算法只针对 TA061 实例获得了最优解.此外,当机器数量等于 5 时,HDCPSO 算法对于大多数问题实例得到的结果均值和最优解一致,而对于其他问题,其均值也非常接近最优解,因此我们说 HDCPSO 算法具有更好的鲁棒性,尤其是当机器数量较少的时候.

表 6 列出了三个算法对于不同规模的测试问题的平均相对偏离度的对比,对于所有问题,HDCPSO 的 ARPD 都是最小的.因此我们说,HDCPSO 求得的解的质量要优于 NPSO 算法,并且更不容易陷入局部极值.

表 6　NEH、NPSO 和 HDCPSO 对于 ARPD 的对比结果

Problem(PS)	NEH	NPSO	HDCPSO
20 × 5	3.393	0.963	0
20 × 10	5.313	2.173	0.327
20 × 20	4.70	1.50	0.217
50 × 5	0.263	0.093	0.003
50 × 10	5.75	3.883	1.52
50 × 20	6.953	4.62	1.92
100 × 5	0.377	0.19	0.009
100 × 10	2.15	1.286	0.533
100 × 20	5.093	4.193	2.14
Average	3.777	2.100	0.741

为了能够更直观的进行 HDCPSO 和 NPSO 算法的比较,我们给出了两种算法对于不同问题的最优解的均值进化曲线的对比,由于篇幅有限,我们只给出了问题 TA31 和 TA51 的进化曲线.以图 2(a)为例,两个算法在执行的初始阶段,收敛速度接近,但是当进化到大于 20 代左右,NPSO 算法就已经停止进化,发生早熟收敛现象.而 HDCPSO 算法尽管也出现了进化停滞现象,但是由于采用 DC 变异和局部搜索策略,降低了早熟收敛发生的可能,从而在大约 300 代收敛到全局最优解.图 2(b)的情况与图 2(a)类似,在此不再赘述.因此,我们可以看出 HDCPSO 算法的收敛速度快于 NPSO 算法,并且解的质量也明显优于 NPSO 算法.

4.2.2　实验对比 2

在本小节中,我们将 HDCPSO 算法与 Kuo 等人最新提出的混合粒子群算法 HPSO[16] 进行比较.HPSO 算法采用基于随机键的实数编码形式,并且通过个体强化策略来试图找到更好的解.

我们将 HDCPSO 算法重新执行,对每个测试问题实例独立执行 10 次,群体大小和最大迭代次数与文献[16]一致,两个算法的对比结果见表 7.

表 7　HPSO 和 HDCPSO 算法的对比

Problem	Size	BKS	HPSO					HDCPSO				
			Min	Max	Avg	Std	ARPD	Min	Max	Avg	Std	ARPD
TA001	20 × 5	1278	1278	1278	1278.0	0	0	1278	1278	1278.0	0	0
TA011	20 × 10	1582	1582	1596	1587.3	4.1	0.34	1582	1593	1585.6	3.66	0.23
TA021	20 × 20	2297	2297	2315	2307.0	5.1	0.44	2297	2310	2304.0	4.55	0.30
TA031	50 × 5	2724	2724	2724	2724.0	0	0	2724	2724	2724.0	0	0
TA041	50 × 10	2991	3034	3063	3053.6	9.2	2.09	3025	3039	3032.5	6.62	1.39
TA051	50 × 20	3850	3923	3963	3944.6	12.2	2.46	3894	3933	3911.6	11.50	1.60
TA061	100 × 5	5493	5493	5493	5493.0	0	0	5493	5493	5493.0	0	0

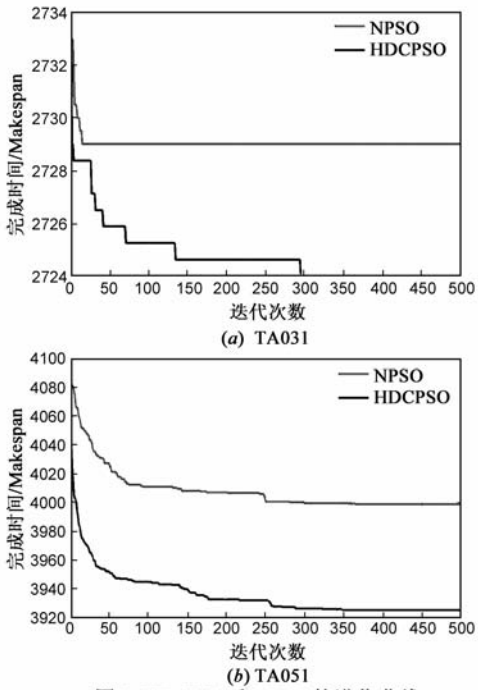

图2 HDCPSO和NPSO的进化曲线

从对比结果中,我们可以看出,对于问题实例TA001、TA031 和 TA061,由于这些问题实例相对简单(机器数远小于作业数量),因此两个算法在每次执行时都能够找到问题最优解.而对于其他问题,HDCPSO 算法无论是最小值、最大值、均值、方差以及平均相对偏离度都是优于 HPSO 算法的.此外,对于小规模问题,如TA011 和 TA021,两个算法的结果相差不多,HDCPSO 算法略好于 HPSO 算法,而对于问题 TA041 和 TA051,HDCPSO 算法获得的解的质量明显优于 HPSO 算法,并且更具有鲁棒性.因此,从整体性能上看,HDCPSO 算法求得解的质量、以及与已知最优解之间的偏离度都优于HPSO 算法.

5 总结

本文针对置换流水车间调度中求解最大完成时间的问题进行了研究,并提出了一种混合的粒子群算法HDCPSO 用于求解 PFSP 中的最小化最大完成时间问题.该算法结合了 Ruiz 等人提出的 IG 算法中的作业毁坏和构造(DC)操作,将 DC 操作用于个体的变异,并且通过个体徘徊来控制变异发生的条件.并且本文采用了基于插入邻域的局部搜索来提高算法的收敛速度.此外,我们采用对群体进行重新初始化的策略,以保证群体多样性,降低早熟现象发生的概率.

通过和 NEH、NPSO 以及最新的 HPSO 算法在不同规模的 Taillard 测试问题上的对比,验证了本文提出的HDCPSO 算法的有效性.本文算法在收敛速度和最优解的质量上,都优于其他三种算法.

本文算法的优势主要体现在 DC 变异、局部搜索以及群体重新初始化,但是 DC 变异和局部搜索通常会消耗一定的时间,所以文本算法在执行效率上比其他三个算法要慢.因此,提高算法的执行效率,减少执行时间值得进一步研究.此外,在本文中,个体徘徊代数是不变的,但是徘徊代数的自适应变化有可能会产生更好的结果,我们将在下一步继续研究这些问题.

参考文献

[1] Garey M R, Johnson D S, Sethi R. The complexity of flowshop and job shop scheduling[J]. Mathematics of Operations Research, 1976, 1(2): 117 – 129.

[2] Eberhart R C, Kennedy J. A new optimizer using particle swarm theory[A]. Proceedings of the Sixth International Symposium on Micro Machine and Human Science[C]. Nagoya, Japan, 1995. 39 – 43.

[3] Tasgetiren M F, Sevkli M, Liang Y C, Gencyilmaz G. Particle swarm optimization algorithm for permutation flowshop sequencing problem[A]. Lecture Notes in Computer Science[C]. Berlin Heidelberg: Springer-Verlag, vol. 3172, 2004. 382 – 389.

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

[5] 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): 3099 – 3111.

[6] Lian Z G, Gu X S, Jiao B. A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan[J]. Applied Mathematics and Computation, 2006, 175(1): 773 – 785.

[7] 周驰,高亮,高海兵. 基于 PSO 的置换流水车间调度算法[J]. 电子学报, 2006, 34(11): 2008 – 2011.
Zhou Chi, Gao Liang, Gao Hai-bing. Particle swarm optimization based algorithm for permutation flow shop scheduling[J]. Acta Electronica Sinica, 2006, 34(11): 2008 – 2011. (in Chinese)

[8] Lian Z G, Gu X S, Jiao B. A novel particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan[J]. Chaos, Solitons and Fractals, 2008, 35(5): 851 – 861.

[9] 张长胜,孙吉贵,欧阳丹彤. 一种自适应离散粒子群算法及其应用研究[J]. 电子学报, 2009, 37(2): 299 – 304.
Zhang Chang-sheng, Sun Ji-gui, OuYang Dan-tong. A self-adaptive discrete particle swarm optimization algorithm[J]. Ac-

ta Electronica Sinica, 2009, 37(2): 299 – 304. (in Chinese)
[10] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem[J]. European Journal of Operational Research, 2007, 177(3): 2033 – 2049.
[11] Nawaz M, Enscore E, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem[J]. Omega, 1983, 11(1): 91 – 95.
[12] Pan Q K, Tasgetiren M F, Liang Y C. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem[J]. Computers & Operations Research, 2008, 35(9): 2807 – 2839.
[13] 张长胜,孙吉贵,杨轻云,郑黎辉. 一种求解车间调度的混合算法[J]. 自动化学报, 2009, 35(3): 332 – 336.
Zhang Chang-sheng, Sun Ji-gui, Yang Qing-yun, Zheng Li-hui. A hybrid algorithm for flowshop scheduling problem[J]. Acta Automatica Sinica, 2009, 35(3): 332 – 336. (in Chinese)
[14] Pan Q K, Tasgetiren M F, Liang Y C. A discrete differential evolution algorithm for the permutation flowshop scheduling problem[J]. Computers & Industrial Engineering, 2008, 55(4): 795 – 816.
[15] Glover F. A template for scatter search and path relinking [A]. Lecture Notes In Computer Science[C]. London, UK: Springer-Verlag, vol. 1363, 1998. 13 – 54.
[16] Kuo I H, Horng S J, Kao T W, et al. An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model[J]. Expert Systems with Applications, 2009, 36(3): 7027 – 7032.

作者简介

田　野　男, 1979 年生于吉林四平. 2006 年毕业于吉林大学软件学院,现为吉林大学计算机科学与技术学院博士研究生. 研究方向为计算智能,数据挖掘.
E-mail: tianye924@yahoo.com.cn

刘大有(通信作者)　男, 1942 年生于河北乐亭. 现为吉林大学计算机科学与技术学院教授、博导. 主要研究方向为知识工程与专家系统、分布智能与多 Agent 系统、不确定性推理、数据挖掘、时空知识表示与推理等.
E-mail: liudy@jlu.edu.cn