文章编号: 2095-6991(2016)04-0001-09

带惩罚费用的非同类机调度问题的粒子群算法

崔倩娜

(云南大学,云南 昆明 650000)

摘要:由于机器处理能力的限制,带惩罚费用的调度问题引起了研究人员的广泛关注. 近年来的研究多数都是近似或最优算法的设计. 粒子群优化算法,因其简单且易于实现等特点,已经应用于各种领域. 针对带有惩罚费用的非同类机调度问题,设计了两种离散的粒子群优化算法,并通过实验比较了两种算法的优劣.

关键词:排序;非同类机器;惩罚费用;离散粒子群优化算法

中图分类号:O226 文献标志码:A

DOI:10.13804/j.cnki.2095-6991.2016.04.001

0 前言

自 Bartal 等人[1]首次提出带惩罚费用的并行机调度问题以来,不同环境下的带惩罚费用的调度问题得到了广泛的研究[2-4]. 但是,多数研究主要是针对并行机的研究,然而在实际生活中不同机器对于工件的处理时间是不同的,所以就产生了对非同类机排序工作的研究,比如黄宇冰等人对此问题做了一些研究[5],他们提出一种基于指派与调度的复合策略的算法[5].

非同类机的问题如下:给定一个独立的工件集 $J=\{J_1,J_2,\cdots,J_n\}$ 和一个机器集 $M=\{M_1,M_2,\cdots,M_n\}$,每个工件应该在其中一台机器完成,或者被所有机器拒绝. 工件 J_j 在机器 M_k 上的处理时间为 p_{jk},工件 J_j 被拒绝产生的损失为 w_j. 一旦工件开始处理,那么这台机器必须将工作做完才可开始下个工作. 同时,每台机器在同一时间内,只能处理一个工件.

引入决策变量 x_{jk},$x_{jk}=1$ 当且仅当工件 J_j 在机器 M_k 处理,带惩罚费用的非同类机调度问题的数学规划形式如下:

$$\begin{cases}\min\left[y+\sum_j w_j(1-z_j)\right],\\ y-\sum_{j=1}^{n}p_{jk}x_{jk}\geqslant 0,\quad k=1,\cdots,m,\\ \sum_{k=1}^{m}x_{jk}=z_j,\ j=1,\cdots,n,\\ x_{jk},z_j\in\{0,1\},j=1,\cdots,n,k=1,\cdots,m,\end{cases}\tag{1}$$

这里 $z_j=1$ 表示工件 J_j 被接受,$z_j=0$ 表示工件 J_j 被拒绝并产生惩罚费用 w_j. 容易验证,最优解中的 y 值就是最大机器完工时间

$$C_{\max}=\max_i\sum_{j=1}^{n}p_{jk}x_{jk}.$$

由于带惩罚费用的非同类机调度问题是同类机调度问题的推广形式,所以此问题是 NP-难[6]的. 非同类机调度问题是否存在近似比小于 2 的近似算法是著名的公开问题. 因此,找到带惩罚费用的非同类机调度问题近似比较好的算法也是较难的. 粒子群算法已经证明了经典的排序问题和其他种类的优化问题[7-9]. 然而,它们都是连续的 PSO,所以需要将连续的 PSO 优化算法转化到调度中,必须采用离散的粒子群优化算法(DPSO). Ali 等研究人员基于模拟退火算法提出了一种优于这种算法的离散粒子群算法[10]来解决平行机器的排序问题;王现鹏等人也提出了一种离散粒子群优化算法[11],两种算法已经被分别研究,并用实例证明了其优越性[10-11]. 本文采用这两种已经被证明具有优越性算法的思想,设计了带惩罚费用的非同类机调度问题的两个粒子群算法.

本文的剩余部分包括:在第二部分,介绍一下 PSO 算法的基本思想;第三部分,介绍两种不同的 DPSO 算法;第四部分给出了实验数据来说明这两种 DPSO 的优劣性;最后,文章给出了总结和未来的研究方向.

1 PSO简介

粒子群算法(PSO)是社会心理学博士 Ken-

收稿日期:2016-04-15

作者简介:崔倩娜(1991-),女,河南林州人,在读硕士,主要从事组合最优化研究. E-mail:cui18211656432@163.com.

nedy[12]和电气工程学博士 Eberhart[12]在1995年受 Reynolds[13]提出的 Boid 模型启发,提出的一种模拟鸟群飞行觅食的仿生优化算法,通过鸟之间的集体协作使群体最优.此算法基于群体迭代,群体在解空间中追随最优粒子进行搜索,其概念简明、实现方便、收敛速度快、参数设置少,是一种高效的搜索算法.

粒子群优化算法是一个非常简单的算法,且能够有效地优化各种函数.从某种程度上说,此算法介于遗传算法和进化规划之间.此算法非常依赖于随机过程,这是和进化规划的相似之处.此算法中朝全局最优和局部最优靠近的调整非常类似于遗传算法中的交叉算子.此算法还采用了适应值的概念,这是所有进化计算方法所共有的特征.

粒子群优化算法是一种基于群智能方法的演化计算技术,在粒子群优化中,每个个体称为一个"粒子",其实每个粒子代表着一个潜在的解.

例如,在一个 D 维的目标搜索空间中,每个粒子看成空间内的一个点.设群体由 n 个粒子构成,也被称为群体规模,过大的 n 会影响算法的运算速度和收敛性.设 $z_j=(z_{j1},z_{j2},\cdots,z_{jD})$ 为第 j 个粒子 $j=(1,2,\cdots,n)$ 的 D 维位置矢量,根据事先设定的适应值函数计算 z_j 当前的适应值,即可衡量粒子位置的优劣;$v_j=(v_{j1},v_{j2},\cdots,v_{jd},\cdots,v_{jD})$ 为粒子 j 的飞行速度,即粒子移动的距离;$p_j=(p_{j1},p_{j2},\cdots,p_{jd},\cdots,p_{jD})$ 为粒子迄今为止搜索到的最优位置;$p_g=(p_{j1},p_{j2},\cdots,p_{jd},\cdots,p_{jD})$ 为整个粒子群迄今为止搜索到的最优位置.

在每次迭代中,粒子根据以下式子更新速度和位置:

$$v_{jd}^{k+1}=wv_{jd}^{k}+c_1r_1(p_{jd}-z_{jd}^{k})+c_2r_2(p_{gd}-z_{pd}^{k}),\tag{2}$$

$$z_{jd}^{k+1}=z_{jd}^{k}+v_{jd}^{k+1},\tag{3}$$

其中,$j=(1,2,\cdots,n)$, $d=1,2,\cdots,D$,k 是迭代次数,r_1,r_2 为[0,1]之间的随机数,这两个参数是用来保持群体的多样性.c_1 和 c_2 为学习因子,其使粒子具有自我总结和向群体中优秀个体学习的能力,从而向自己的历史最优点以及群体内历史最优点靠近.这两个参数对粒子群算法的收敛起的作用不是很大,但如果适当调整这两个参数,可以减少局部最优的困扰,当然也会使收敛速度变快.w 是惯性权重,起着权衡局部最优能力和全局最优能力的作用.

本文将会介绍与经典 PSO 相似的结构,包括 PSO 的主要特征,我们的算法将会解决平行机器排序问题.

2 离散粒子群优化算法

2.1 机器排序的 DPSO

Ali[10]等人在平行机器并联工作的排序算法上提出一种离散粒子群优化算法.根据这个思想,本文用相似的方式来解决非同类机器的排序问题.用 n 维代表 n 项工作,因此,可以对这一序列工作安排机器或者安排被拒绝,而且其长度与工作的数量相同.第 d 个粒子的位置代表第 d 项工作被安排的情况.表1展示了5项工作和2台机器排序情况的一个例子,我们用3来代表这项工作被拒绝.

表1 给工作安排机器的初始值

工作1	工作2	工作3	工作4	工作5
1	2	2	3	1

粒子按照下式更新粒子速度和位置:

$$V_k^{t+1}=V_k^t\bar{O}\left(R_1\overset{\times}{O}(P_k^t\bar{O}X_k^t)\right)\overset{+}{O}\left(R_2\overset{\times}{O}(P_g^t\bar{O}X_k^t)\right),\tag{4}$$

$$X_k^{t+1}=X_k^t\bar{O}V_k^{t+1},\tag{5}$$

其中,R_1,R_2 为一列 n 维随机数,只包含0和1两个元素,而且随机数列是由贝努里分布产生的,产生1的概率为0.3.然而,合适的位置应该通过考虑问题的数据来进行调整.V_k^t,X_k^t 为第 k 个粒子的速度和位置,其初始值是随机产生的一组整数列,P_k^t,P_g^t 为第 k 个粒子的个体最优位置和全局最优位置.定义的运算按照下面的规则.

减法操作($\bar{O}$):通过 n 维向量可以显示出第 k 个粒子当前位置和最优位置 P_k^t(或者 P_g^t)的不同.向量 X_k^t 中的每个元素可以与获得的最优向量中对应元素做比较,比较元素是否相等.如果对应元素相等,则忽略这项运算.如表2所列.

乘法操作$\left(\overset{\times}{O}\right)$:通过这项操作,我们能增强研究 DPSO 系统的能力.增强这项能力是通过对两个不同的矢量做乘法运算,然后产生不同的二进制矢量.这些随机的二进制矢量在实值的 PSO 中表示出随机的工作数目.这个乘法操作与阿达玛积是等价的,两个 n 维行向量 $\boldsymbol{A}$,$\boldsymbol{B}$ 的阿达玛积是 $\boldsymbol{A}\cdot\boldsymbol{B}=a_ib_i$,是一个 n 维行向量.表3是一个阿达

玛积的例子.

表 2　DPSO1 减法操作

	工作 1	工作 2	工作 3	工作 4	工作 5
$A(P_g^t)$	1	2	2	3	1
$B(X_k^t)$	1	1	2	2	1
$A\bar{O}B$	0	2	0	3	0

表 3　DPSO1 乘法操作

	工作 1	工作 2	工作 3	工作 4	工作 5
A	0	1	0	1	1
B	1	1	2	3	1
$A\overset{\times}{O}B$	0	1	0	3	1

加法操作$\left(\overset{+}{O}\right)$:这个操作采用了遗传算法里的交叉操作思想. 对于交叉操作,我们从粒子链中随机选择了两个关键点作为交叉点,随机选择交叉点前或者后为交叉位置,将对应粒子要进行交叉的元素进行交叉互换,其余元素保持不变,从而生成两个新的向量,然后再在获得的两个新粒子中随机选取一个作为这次加法操作的结果. 表 4 是交叉操作的一个例子.

表 4　DPSO1 加法操作

	工作 1	工作 2	工作 3	工作 4	工作 5
A	1	2	3	1	1
B	2	1	3	3	1
A^*	2	2	3	3	1
B^*	1	1	3	1	1
$A\overset{+}{O}B$	1	1	3	1	1

可以看出来,提出的 DPSO 保留了 PSO 的基本原理,只有惯性权重是一个矢量. 下面是 DPSO 的算法,用 $f(X)$ 来表示工作运行时间与被拒绝费用之和最小化.

Begin

t=0;

For k: 1 to swarm size

$X_k^t \leftarrow$Generate a particle at random;

$P_k^t \leftarrow X_k^t$;

End for

$P_g^t \leftarrow \left\{ X_l^t \middle| l=\arg\min_{\forall k}\{f(X_k^t)\} \right\}$;

Do

For k: 1 to swarm size

$V_k^{t+1} \leftarrow$Update the k^{th} particle velocity vector u sin g (4);

$X_k^{t+1} \leftarrow$Update the k^{th} particle position vector u sin g (5);

If $f(X_k^{t+1}) < f(P_k^t)$

$P_k^{t+1} \leftarrow X_k^{t+1}$;

Else

$P_k^{t+1} \leftarrow P_k^t$;

End if

End for

If $f(P_g^t) > \min_{\forall k}\{f(P_k^{t+1})\}$;

$P_g^{t+1} \leftarrow \left\{ P_l^{t+1} \middle| l=\arg\min_{\forall k}\{f(P_k^{t+1})\} \right\}$;

End if

$t \leftarrow t+1$;

Until stopping criteria are true.

End

2.2　工作排序的 DPSO

在研究带有阻塞的排列调度问题中,WANG X P 等人[12] 提出一种离散粒子群算法,采用他们的思想,我们针对本文的问题,也使用这个算法来解决. 一个 m 维序列代表 m 台机器,因此,可以对 $m+1$ 维序列机器安排工作. 第 d 个粒子的位置代表第 d 台机器被安排的工作情况,安排在第 $m+1$ 维的工作,表示此工作被拒绝.

2.2.1　**种群初始化**　由 n 个粒子组成的粒子群 P 可以由两种方式产生,以保证解决方案的质量. 第一种解决方式是 NEH 启发式[14],另一种方式是随机生成的群体. 简单地描述一下 NEH 启发式方式. 一组没有排序的工作序列按照其被所有机器加工所需要的总时间来进行排序. 设包含工作的序列为

$$S=(s(1),s(2),\cdots,s(n)).$$

先建立一个工作序列空集 π,首先将前两项工作 $s(1),s(2)$ 插入 π 中,并且按照工作被机器完成的时间总和来排序,得到 $\pi=(s(1),s(2))$ 或者 $\pi=(s(2),s(1))$. 再插入第 3 项工作 $s(3)$,同样依照工作被所有机器来完成的时间总和与 $s(1),s(2)$ 做比较,然后确定其插入位置. 依照这种方式,得到一个工作序列

$$\pi=(s(1),s(2),\cdots,s(n)).$$

2.2.2　**速度更新方式**　由于设计速度更新的模型可以让算法实施起来更有效果,更简单,所以我们使用下面的更新方式来更新速度:

$$V_k^{t+1} = (c_1 \times V_k^t) + (c_2 \times (P_i^t - X_i^t)) + (c_3 \times (G^t - X_i^t)), \quad (6)$$

其中,c_1,c_2,c_3 是在(0,1)内预先产生的随机数.与依靠位移来产生速度的方式不同,这个DPSO产生的速度是一个 m 维行向量,其中包括0元素和被安排的工作元素.这项操作中的运算是依照下面的方式进行.

(1)减法操作(—).

这项操作是决定于第 i 个粒子的当前位置 X_i^t 和最优位置 P_i^t(或者 G^t),如果第 i 个粒子的 X_i^t 和 P_i^t(或者 G^t)相同,则获得的新序列中第 i 个元素的值为0,否则,就取 P_i^t(或者 G^t)中对应元素的值.表5举例说明了减法操作(—).注意,如果 X_i^t 和 P_i^t(或者 G^t)完全一样,则所获得的序列是空集,并且在速度的决定公式里可以被忽略.

表5 DPSO2减法操作

机	1	2	3	4	5	6	7	8	9	10
X_i^t	4	8	1	5	7	2	6	10	3	9
P_i^t	4	7	1	5	8	9	2	10	3	6
$P_i^t - X_i^t$	0	7	0	0	8	9	2	0	0	6

(2)乘法操作(×).

乘法运算可以增加DPSO的搜索能力.用 a 来表示在 V_i^t(或者 $(P_i^t - X_i^t)$ 或 $(G^t - X_i^t)$)中非零元素的个数,我们从 V_i^t(或者 $(P_i^t - X_i^t)$ 或 $(G^t - X_i^t)$)中随机选取$[ac_1]$(或$[ac_2]$或$[ac_3]$)个不为0的元素,表6是展示乘法操作(+)的一个例子.

表6 DPSO2乘法操作

机 器	1	2	3	4	5	6	7	8	9	10
$P_i^t - X_i^t$	0	7	0	0	8	9	2	0	0	6
$c_2 \times (P_i^t - X_i^t)$	0	7	0	0	0	9	0	0	0	6

(3)加法操作(+).

加法操作是用来将3个 m 维行向量结合为1个行向量的操作,用 $\pi_k = (\pi_k(1), \pi_k(2), \cdots, \pi_k(n))$,$k = 1,2,3$ 来表示第 k 个序列,并且 $\pi_k(j)$ 表示被安排在 j 位置的工作.然后依照下面描述的结合方式来操作.

第一步:令 $j = 1$,先建立一个0序列 $\pi_k = (\pi(1), \pi(2), \cdots, \pi(n))$;

第二步:对于 π_k,先检验 π_k 与 π 中的元素是否吻合,如果相同,则令 $\pi_k(j) = 0$,这一步是为了确定序列 π 中没有复制的元素;

第三步:依照下面方式决定 $\pi(j)$ 的值:

(1)如果 $\pi_k(j) = 0$,$k = 1,2,3$,那么令 $\pi(j) = 0$;

(2)如果只有一个元素 $\pi_k(j) > 0$,那么令 $\pi(j) = \pi_k(j)$;

(3)如果有两个元素大于0,比如 $\pi_1(j) > 0$,$\pi_2(j) > 0$,那么,先产生一个(0,1)内的随机数 r,如果

$$r < \frac{c_1}{(c_2 + c_3)},$$

则令 $\pi(j) = \pi_1(j)$,否则,令 $\pi(j) = \pi_2(j)$;

(4)如果3个元素都大于0,则先产生一个(0,1)内的随机数 r,如果

$$r < \frac{c_1}{(c_1 + c_2 + c_3)},$$

令 $\pi(j) = \pi_1(j)$;如果

$$\frac{c_1}{(c_1 + c_2 + c_3)} < r < \frac{(c_1 + c_2)}{(c_1 + c_2 + c_3)},$$

那么令 $\pi(j) = \pi_2(j)$;如果

$$r > \frac{(c_1 + c_2)}{(c_1 + c_2 + c_3)},$$

那么令 $\pi(j) = \pi_3(j)$.

第四步:令 $j = j + 1$,如果 $j > n$,就停止,否则,返回第二步.

表7举例说明了加法操作(+).在这个例子中,最后结果中第3和第4的位置是0,这是由于这两个位置上的工作已经被安排过了.

表7 DPSO2速度加法操作

机 器	1	2	3	4	5	6	7	8	9	10
π_1	0	1	0	0	2	0	7	0	0	3
π_2	4	0	0	3	0	0	0	5	0	0
π_3	0	3	0	4	2	0	7	6	0	0
$\pi_1 + \pi_2 + \pi_3$	4	3	0	0	2	0	7	5	0	0

2.2.3 位置更新方式 在速度更新之后,更新位置,采用下列方式:

$$X_i^{t+1} = X_i^t \oplus V_i^{t+1}. \quad (7)$$

加法操作(⊕):

这项操作包括两个步骤.第一步,在 V_i^t 中包含的工作,如果在 X_i^t 中也存在,则将 X_i^t 中的这些工作移除;第二步,将 X_i^t 中的工作依次插入到 V_i^t 中没有被安排工作,即0元素的位置上.表8是这项操作的例子,更清楚地展示出这项操作的具体

步骤.

表 8　DPSO2 位置加法操作

机器	1	2	3	4	5	6	7	8	9	10
X_i^t	4	8	1	5	7	2	6	10	3	9
V_i^t	0	0	0	8	0	9	0	0	1	0
$X_i^t \oplus V_i^t$	4	5	7	8	2	9	6	10	1	3

算法的主程序与 2.1 中 DPSO 的主程序相同,只是将迭代公式中(4)、(5)分别换为(6)、(7).

3　实验比较

通过解决大量问题,我们对两种 DPSO 进行实验对比,我们采用 Lee 等[15] 使用过的被 Gupta 和 Ruiz 等[16-17] 设计的实验框架来测试程序,命名为 $E1$,$E2$,$E3$ 的三个结构是对比了四个有影响力的变量:机器数量(m),工作数量(n),工作完成时间矩阵 P 的范围,和每项工作被拒绝之后的惩罚费用(W). 表 9 显示了所有实验结构的总结.

表 9　实验结构

	m	n	P	W
E1	3,4,5	2m　3m　5m	U(1,20) U(20,50)	U(0,1) U(0,3)
E2	3,6,8,10	30,50,100	U(50,100)	U(0,3) U(0,5)
E3	3	4m+1	U(1,20) U(20,50)	U(0,1) U(0,3)
	5	3m+1	U(50,100) U(100,200)	U(0,5)

实验结构 $E1$,机器数量(m)采用 3 个标准:3,4,5;工作数量也采用 3 个标准:$2m$,$3m$,$5m$;完成时间和惩罚费用都从离散的范围 $U(.,.)$ 中分配,而且采用 2 个标准:$U(1,20)$ 和 $U(20,50)$;惩罚费用采用两个标准:$U(0,1)$,$U(0,3)$. 实验结构 $E2$,机器数量采用 4 个标准:3,6,8,10;工作数量设立 3 个标准:30,50,100;工作时间仅仅在 $U(50,100)$ 中产生;惩罚费用在 2 个范围内产生:$U(0,3)$,$U(0,5)$. 实验 $E3$,机器和工作数量是一定的;工作时间分别在 4 个范围中产生;惩罚费用分别在 3 个范围内产生.

两种算法都用 MATLAB 来实现,每个不同数据都运行 100 次来产生最优结果. 每个算法的效果,用两种衡量方式,即最后的最优结果和运行 100 次所需要的时间,其中 z 代表最后的优化结果,time 代表迭代 100 次需要的总时间.

实验 $E1$ 的结果如表 10 所列,可以很清晰地看出所有不同资料运行时间中 DPSO 2 的时间要比 DPSO 1 的运行时间少很多. 但是这并不意味

表 10　实验 *E1* 对比结果

				DPSO1		DPSO2	
m	n	P	W	z	time	z	time
3	6	$U(1,20)$	$U(0,1)$	8.13	2.978 0	9.69	0.690 0
	9			4.71	3.443 0	12.81	0.745 0
	15			2.42	4.293 0	8.33	0.702 0
	6	$U(20,50)$		1.63	3.017 0	77.39	0.703 0
	9			25.12	3.416 0	71.42	0.692 0
	15			20.94	4.369 0	67.98	0.680 0
	6	$U(1,20)$	$U(0,3)$	5.14	3.662 0	12.88	0.701 0
	9			6.07	4.269 0	11.24	0.714 0
	15			10.37	5.348 0	5.04	0.707 0
	6	$U(20,50)$		3.37	3.646 0	73.75	0.736 0
	9			29.89	4.148 0	79.96	0.717 0
	15			27.64	5.408 0	73.83	0.739 0
	15			5.14	3.662 0	12.88	0.701 0

续表 10

				DPSO1		DPSO2	
m	n	P	W	z	time	z	time
4	6	$U(1,20)$	$U(0,1)$	15.93	2.972 0	21.13	0.757 0
	9			9.74	3.430 0	13.11	0.778 0
	15			5.81	4.314 0	18.23	0.786 0
	6	$U(20,50)$		63.06	2.958 0	107.07	0.783 0
	9			61.82	3.414 0	116.48	0.785 0
	15			50.66	4.346 0	94.67	0.788 0
	6	$U(1,20)$	$U(0,3)$	15.91	4.034 0	15.05	0.764 0
	9			13.76	4.830 0	19.85	0.773 0
	15			17.85	6.329 0	17.33	0.761 0
	6	$U(20,50)$		53.69	4.087 0	121.57	0.760 0
	9			28.43	4.881 0	113.42	0.789 0
	15			66.35	6.257 0	89.44	0.785 0
	15			15.91	4.034 0	15.05	0.764 0
5	6	$U(1,20)$	$U(0,1)$	20.55	2.976 0	24.63	0.812 0
	9			24.27	3.538 0	25.83	0.815 0
	15			19.06	4.321 0	25.57	0.838 0
	6	$U(20,50)$		100.56	2.978 0	119.24	0.787 0
	9			80.23	3.565 0	134.27	0.837 0
	15			37.68	4.303 0	137.70	0.849 0
	6	$U(1,20)$	$U(0,3)$	25.97	4.613 0	31.76	0.845 0
	9			23.74	5.488 0	17.26	0.844 0
	15			34.47	7.296 0	28.02	0.847 0
	6	$U(20,50)$		46.15	3.555 0	145.25	0.862 0
	9			89.00	4.220 0	125.74	0.870 0
	15			71.29	5.989 0	117.25	0.827 0

着,DPSO 2 就是比较好的算法,我们观察 z 值的变化,在工作时间的取值范围增加时,DPSO 2 的波动非常大. 比如,机器数量为 5 时,不管惩罚费用的取值怎样变化,只要时间取值增大时,每个工作数量 z 值都会增加,但是 DPSO 2 增加极大,尤其是工作数量为 9,惩罚费用在 $U(0,3)$ 之间时,DPSO 2 的 z 值增加了 108.48,而 DPSO 1 的 z 值增加 65.26,相比之下,DPSO 1 就稳定许多.

实验 $E2$ 正好可以减少 P 的不稳定性因素造成的波动,其实验结果如表 11 所列. DPSO 1 的运行时间随着工作数增加而增加,在工作数量达到 100 时,DPSO 1 的运行时间在 50 秒左右,需要等待很长时间,相比之下,DPSO 1 的运行时间就很少,而且在工作数量增加的时候,运行时间也不会增加. DPSO 2 的运行时间是在机器数量增加的时候才会有明显的增加,但是波动也不会太大,比如在工作数量都是 100,惩罚费用都在 $U(0,3)$ 之间时,机器数量为 3 和 6 的时候,运行时间增加了 0.237 0 秒.

实验 $E3$ 是在机器和工作数量一定的条件下,测试工作完成时间和被拒绝后的惩罚费用对算法的影响,实验结果如表 12 所列. 数据表明在惩罚费用变化的时候,两种算法的运行时间没有变化,即算法的运行速度不受惩罚费用的影响;在工作完成时间改变的时候,两种算法的运行时间基本保持不变,即算法的运行速度也不受时间的影响.

上面的数据表明,两种算法的运行速度只受工作数量和机器数量的影响,而不受工作完成时间和惩罚费用的影响. 数据显示,在数据变大时,DPSO 2 的运行速度比 DPSO 1 的运行速度要快,

表 11　实验 *E*2 对比结果

				DPSO1		DPSO2	
m	*n*	*P*	*W*	*z*	time	*z*	time
3	30	*U*(50,100)	*U*(0,3)	80.59	7.266 0	189.01	0.731 0
	50			79.43	15.199	169.12	0.713 0
	100			4.76	49.434	172.17	0.676 0
	30		*U*(0,5)	65.42	6.691 0	168.35	0.745 0
	50			92.32	17.611	172.19	0.748 0
	100			9.40	49.890	169.30	0.738 0
6	30	*U*(50,100)	*U*(0,3)	255.30	6.644 0	358.80	0.890 0
	50			192.66	15.167	338.55	0.919 0
	100			208.10	52.066	355.59	0.913 0
	30		*U*(0,5)	217.75	7.447 0	399.35	0.906 0
	50			288.19	15.225	376.36	0.889 0
	100			317.61	48.542	377.48	0.883 0
8	30	*U*(50,100)	*U*(0,3)	375.31	7.443 0	499.79	1.004 0
	50			383.80	15.187	508.08	0.983 0
	100			370.28	45.388	516.09	1.044 0
	30		*U*(0,5)	423.18	6.876 0	520.02	1.032 0
	50			372.77	15.920	526.74	1.033 0
	100			362.72	51.746	478.87	0.999 0
10	30	*U*(50,100)	*U*(0,3)	481.67	7.789 0	638.06	1.142 0
	50			543.87	15.425	620.19	1.119 0
	100			538.82	49.692	690.38	1.023 0
	30		*U*(0,5)	515.96	7.327 0	667.84	1.036 0
	50			525.35	16.900	610.22	1.142 0
	100			567.41	48.930	645.15	1.130 0

表 12　实验 *E*3 对比结果

				DPSO1		DPSO2	
m	*n*	*P*	*W*	*z*	time	*z*	time
3	13	*U*(1,20)	*U*(0,1)	13.90	4.447 0	12.15	0.726 0
			U(0,3)	8.47	4.512 0	9.18	0.725 0
			U(0,5)	16.43	4.592 0	11.63	0.739 0
		U(20,50)	*U*(0,1)	1.40	4.058 0	81.65	0.717 0
			U(0,3)	4.34	4.026 0	77.47	0.740 0
			U(0,5)	6.67	4.023 0	87.67	0.712 0
		U(50,100)	*U*(0,1)	54.85	4.310 0	178.46	0.727 0
			U(0,3)	64.10	4.034 0	174.55	0.672 0
			U(0,5)	8.23	4.709 0	164.60	0.720 0
		U(100,200)	*U*(0,1)	107.21	4.400 0	346.99	0.751 0
			U(0,3)	107.22	4.061 0	328.21	0.727 0
			U(0,5)	105.51	4.037 0	344.71	0.743 0

续表 12

				DPSO1		DPSO2	
m	n	P	W	z	time	z	time
5	16	$U(1,20)$	$U(0,1)$	26.61	4.533 0	25.10	0.857 0
			$U(0,3)$	18.62	4.467 0	29.90	0.810 0
			$U(0,5)$	24.55	5.084 0	24.79	0.848 0
		$U(20,50)$	$U(0,1)$	68.48	4.504 0	133.22	0.859 0
			$U(0,3)$	47.73	4.465 0	120.39	0.863 0
			$U(0,5)$	84.09	5.234 0	140.64	0.809 0
		$U(50,100)$	$U(0,1)$	189.27	4.477 0	306.54	0.854 0
			$U(0,3)$	171.36	4.515 0	308.22	0.843 0
			$U(0,5)$	171.66	4.480 0	294.40	0.832 0
		$U(100,200)$	$U(0,1)$	396.14	4.540 0	566.76	0.882 0
			$U(0,3)$	318.16	4.529 0	625.30	0.850 0
			$U(0,5)$	305.08	5.184 0	626.75	0.852 0

所以 DPSO2 的收敛速度更快,适合用于解决大数据的机器排序问题.

4 总结

为了研究带有惩罚费用非同类机器的排序问题,本文将基本的 PSO 进行改进的两种 DPSO 算法,均在其运算方式上作出改变,不再是古老的"+,-,×",并且在 MATLAB 的实现中得出一些实验结果,并分析了两种优化算法的优劣性,得出第二种优化方法的更好结果.所以,在实际生活中采用第二种离散粒子群优化算法去解决问题,将会达到更快更精确的结果.但是,第二种离散粒子群优化算法并不是最好的方法,它不能够求出问题的确切值,只能使结果接近确切值,所以未来需要更努力地去研究离散粒子群优化算法.

参考文献:

[1] BARTAL Y, LEONARDI S, SPACCAMELA A M, et al. Multiprocessor Scheduling with rejection[J]. SIAM J Discrete Math,2000,13(1):64-78.

[2] SHABTAY D, GASPAR N, KASPI M. A survey on offline scheduling with rejection [J]. Journal of Scheduling,2013,16(1):3-28.

[3] QU J W, ZHONG X L, WANG G Q. Animproved heuristic for parallel machine scheduling with rejection[J]. European Journal of Operational Research, 2015(241):653-661.

[4] LI W, LI J, ZHANG X, et al. Penalty cost constrained identical parallel machine scheduling problem [J]. Theoretical Computer Science,2015(607):181-192.

[5] 黄宇冰,刘建峰,赵良才.基于复合策略的平行非同类机调度问题研究[J].计算机应用,2006,26(11):2643-2644.

[6] ADAMS J, BALAS E, ZAWACK D. The shifting bottleneck procedure for job-shop scheduling [J]. Management Sciences,1988,34(3):391-401.

[7] TASGETIREN M F, LIANG Y C, SEVKLI M. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem[J]. Eur J Oper Res, 2007,177(3):1930-1947.

[8] EI-ABD M, HASSAN H, ANIS M, et al. Discrete cooperative particle swarm optimization for FPGA placement[J]. Applied Soft Computing,2010,10(1): 284-295.

[9] MANDAL K K, BASU M, CHAKRABORTY N. Particle swarm optimization teachnique based short-term hydrothermal scheduling[J]. Applied Soft Computing,2008,8(4):1392-1399.

[10] KASHAN A H, KARIMI B. A discrete particle swarm optimization algorithm for scheduling parallel machines[J]. Computers & Industrial Engineering, 2009,56(1):216-223.

[11] WANG X P, TANG L X. A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking[J]. Applied Soft Computing,2012,12(2): 652-662.

[12] KENNEDY J, EBERHART R C. Particle swarm

optimization[C]. Proceeding of IEEEInternational Conference on Neural Networks, New Jersey: Piscataway,1995:1942-1948.

[13] REYNOLDS C W. Flocks, herds and schools: A distributed behavioral model[J]. ACM SIGGRAPH Computer Graphic,1987,21(4):25-34.

[14] NAWAZ M, ENSCORE E E, HAM I. Aheuristic algorithm for them-machine, n-job flow-shop squencing problem[J]. Omega,1983,11(1):91-95.

[15] LEE W C, WU C C, CHEN P. A simulated annealing approach to makespan minimization on identical parallel machines[J]. The International Journal of Advanced Manufacturing Technology, 2006, 31(3-4):328-334.

[16] GUPTA J N D, RUIZ-TORRES A J. A LISTFIT heuristic for minimizing makespan on identical parallel machines[J]. Production Planning & Control, 2001,12(1):28-36.

[17] LEE C Y, MASSEY J D. Multiprocessor scheduling: Combining LPT and MULTIFIT[J]. Discrete Applied Mathematics,1988,20(3):233-242.

[责任编辑:赵慧霞]

Particle Swarm Optimization Algorithm with Penalty Cost for Non-identical Machine Scheduling Problem

CUI Qian-na

(Yunnan University, Kunming 650000, China)

Abstract: Due to the limitations of the machine processing, scheduling problem with penalty cost caused extensive concern of the researchers. In recent years, most of the similar studies are based on the design of approximate or optimal algorithm. Meanwhile, this paper designs two kinds of discrete particle swarm optimization algorithm for non-identical machine scheduling problem with penalty cost, because the particle swarm optimization algorithm is simple and easy to achieve and has been used in various fields. Furthermore, this study compares the pros and cons of the two algorithms through experiment.

Key words: scheduling; non-identical machines; penalty cost; discrete particle swarm optimization algorithm