求解车间调度问题的自适应混合粒子群算法

张长胜[1)] 孙吉贵[2)] 欧阳丹彤[2)] 张永刚[2)]

[1)](东北大学信息科学与工程学院 沈阳 110004)

[2)](吉林大学计算机科学与技术学院符号计算与知识工程教育部重点实验室 长春 130012)

摘 要 针对最小完工时间的流水车间作业调度问题,提出了一种自适应混合粒子群进化算法——AHPSO,将遗传操作有效地结合到粒子群算法中.定义了粒子相似度及粒子能量,粒子相似度阈值随迭代次数动态自适应变化,而粒子能量阈值与群体进化程度及其自身进化速度相关.此外,针对算法运行后期进化速度慢的缺点,提出了一种基于邻域的随机贪心策略进一步提高算法的性能.最后将此算法在不同规模的实例上进行了测试,并与其他几种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几种算法,并且能够有效求解大规模车间作业问题.

关键词 粒子群优化;车间调度;粒子相似度;粒子能量;贪心策略

中图法分类号 TP301 **DOI 号**: 10.3724/SP.J.1016.2009.02137

A Self-Adaptive Hybrid Particle Swarm Optimization Algorithm for Flow Shop Scheduling Problem

ZHANG Chang-Sheng[1)] SUN Ji-Gui[2)] OUYANG Dan-Tong[2)] ZHANG Yong-Gang[2)]

[1)](*School of Information Science and Engineering, Northeastern University, Shenyang* 110004)

[2)](*Key Laboratory of Symbol Computation and Knowledge Engineering of the Ministry of Education, College of Computer Science & Technology, Jilin University, Changchun* 130012)

Abstract A hybrid self-adaptive algorithm is proposed to solve the flow shop scheduling problem with the objective of minimizing makespan, which combined the particle swarm optimization algorithm and genetic operators together. The particle similarity and particle energy are defined. The threshold of particle similarity dynamically changes with iterations and the particle energy depends on the swarm evolving degree and the particle's evolving speed. In order to improve the proposed algorithm performance further, a neighborhood based random greedy search strategy is introduced to overcome the shortcoming of evolving slowly in the later running phase. Finally, the proposed algorithm is tested on different scale benchmarks and compared with the recently proposed efficient algorithms. The result shows that the solution quality and the stability of the HPGA both precede the other two algorithms. It can be used to solve large scale flow shop scheduling problem.

Keywords particle swarm optimization; flow shop scheduling; particle similarity; particle energy; greedy strategy

---

收稿日期:2007-10-18;最终修改稿收到日期:2008-12-10.本课题得到国家自然科学基金重大项目(60496320,60496321)、国家自然科学基金(60773097,60873148)、新世纪优秀人才支持计划项目基金、吉林省科技发展计划项目基金(20060532,20080107)、吉林省青年科研基金(20080107,20080617)及东北师范大学自然科学青年基金(20081003)资助.**张长胜**,男,1980 年生,博士研究生,研究方向为智能信息处理. 孙吉贵,男,1962 年生,教授,博士生导师,研究领域为自动推理、约束程序. **欧阳丹彤**(通信作者),女,1968 年生,教授,博士生导师,研究领域为基于模型的诊断. E-mail: ouyd@jlu.edu.cn. **张永刚**,男,1975 年生,博士,讲师,研究方向为约束程序.

1 引 言

调度问题是很多实际流水线生产调度问题的简化模型,因此其研究具有极高的理论价值和实践价值.本文研究的置换流水车间作业调度问题是在满足工件约束和机器约束条件下,使得最小完工时间尽可能小.工件约束指每个工件在每台机器上恰好加工一次,每个工件在每个机器上的加工顺序相同;机器约束指每台机器在任何时刻至多加工一个工件,每台机器加工的各工件的顺序相同.该问题一般可以描述为:n 个待加工的作业 $J=\{J_1,J_2,\cdots,J_n\}$,需要在 m 台机器上加工 $M=\{M_1,M_2,\cdots,M_m\}$,每个作业包含 m 道工序 $J_i=\{O_{i,1},O_{i,2},\cdots O_{i,m}\}$,其中 $O_{i,k}$ 代表作业 i 在机器 k 上的加工时间为 t_{ik}($1\leq i\leq n,1\leq k\leq m$)的工序.作业 J_j 的完工时间 C_j 为其最后一个工序完成时间即 $C_j=C_{jm}$.求解目标是求得一个可行调度,使得加工完所有作业所花的时间 $C_{\max}$ 尽可能少.该问题可用如下的数学模型表示:

$$C_{\pi_1 1}=t_{\pi_1 1}$$

$$C_{\pi_j 1}=C_{\pi_{j-1}1}+t_{\pi_j 1},\ \forall j\in\{2,\cdots,n\}$$

$$C_{\pi_1 k}=C_{\pi_1 k-1}+t_{\pi_1 k},\ \forall k\in\{2,\cdots,m\}$$

$$C_{\pi_j k}=\max\{C_{\pi_{j-1},k},C_{\pi_j k-1}\}+t_{\pi_j k},\quad \forall j\in\{2,\cdots,n\},\ k\in\{2,\cdots,m\}$$

$$C_{\max}=\max C_j,$$

其中,C_{jk} 表示工序 O_{jk} 的完工时间.

此问题已被证明是 NP 难度问题[1],因此,精确方法[2]在合理的时间内只能求解小规模问题,其求解时间随着问题规模成指数倍增长.而启发式算法能够在可接受的时间内,使用较少的存储空间求得问题近似最优解或最优解,主要分为构造启发式[3]和元启发式两种[4]:构造启发式方法虽可以在较短时间内获得调度问题的解,但其在构造调度的过程中依赖根据问题局部信息设计的调度规则,所获得的调度一般为局部最优解;元启发式方法,是基于仿生学机理的调度算法能够在可行时间内以较大概率获得该类问题的最优解或近似最优解,成为求解各种车间调度问题的有效算法,正受到研究者的广泛关注.

粒子群算法(PSO)是受鸟群觅食启发提出的一种进化计算方法,其收敛速度快、易于实现,被成功应用在多个领域中[5].目前,应用 PSO 算法求解调度问题的研究还很少,实验表明,在求解调度问题时,它们较 GA 算法更为有效.但已提出的算法都存在早熟收敛、易陷入局部最优、进化后期算法收敛速度明显下降等缺点[6-7],主要是由于进化过程中粒子能量不断下降,导致粒子进化停滞不前,群体多样性过低造成的.为了克服这些不足,本文提出了一种混合元启发式算法-AHPSO,将 PSO 算法与 GA 算法[8]结合在一起,利用遗传操作不断引入新的信息指导群体的进化.定义了粒子相似度及粒子能量,粒子相似度阈值随迭代次数动态自适应变化,而粒子能量阈值与群体进化程度及其自身进化速度相关.使用排序策略保持群体的多样性,当相邻的两个粒子的相似度大于其当前的相似度阈值时,对其中的一个粒子执行变异操作.设计了一种基于遍历矩阵的快速计算最小完工时间方法.此外,针对进化后期进化速度慢的缺点,提出了一种基于邻域的随机贪心策略,进一步提高算法的性能.最后,分析了算法的复杂度及收敛性,并通过实验对比证明了算法的有效性.

2 AHPSO 算法

为了使用 PSO 算法求解调度问题,Rameshkumar 提出了一种置换离散粒子群算法[7],粒子在更新过程中只交换相应位置的元素,而不引入新元素.受其启发,我们将置换的思想引入到所提出的算法中.对于一个含有 n 个作业的流水调度问题,粒子 i 的位置及速度均被表示为一个满足"alldifferent"约束[9]的 n 维向量,即所有作业的一个全排列,然后结合 GA 算法中的交叉及变异操作不断更新粒子的位置及速度.

2.1 算法进化模型

在 PSO 算法中,每个粒子的行为主要受其当前动量项、个体认知部分及群体认知部分的影响.因此,传统的粒子速度公式可通过将粒子的当前速度与其个体最优解及当前的群体最优解分别进行交叉来取代,粒子的位置更新也相应地变为将粒子当前位置与当前速度交叉求得.粒子速度及位置更新公式可表示如下:

$$V_i(k+1)=V_i(k)\otimes P_{\text{gbest}}\otimes P_{\text{ibest}} \quad (1)$$

$$X_i(k+1)=X_i(k)\otimes V_i(k+1) \quad (2)$$

其中符号$\otimes$表示交叉操作.由上面的公式可以看出,每个粒子追随其当前个体最优解及全局最优解运动.与传统的 PSO 算法一样,它具有快速收敛、计算

简单等优点.但是,进化过程中粒子的速度会迅速逼近零,即粒子的当前速度和其当前位置相同,使算法易于陷入局部最优解,如实验部分的 AHPSO-S-A 算法所示.为此,本文引入了粒子能量及粒子能量阈值的概念.粒子能量阈值随着迭代次数粒子的进化速度动态自适应调整,使算法在进化初期具有较强的全局搜索能力,在后期则侧重局部精化能力.

定义 1.　给定粒子 P_i,其当前位置和速度分别为 X_i,V_i,当前的个体最优位置和群体最优位置分别为 P_{ibest},P_{gbest}.此粒子当前所具有的能量可计算如下:

$$energy(P_i)=\Big(0.6\times\sum_{j=1}^{dim}same(P_{\text{ibest}}(j),P_{\text{gbest}}(j))+1.4\times\sum_{k=1}^{dim}same(X_i(k),V_i(k))\Big)/(2\times dim),$$

其中,$same(x,y)=\begin{cases}0, & \text{若 } x=y\\ 1, & \text{若 } x\neq y\end{cases}$.

粒子能量用于刻画粒子的搜索能力,与粒子当前状态及群体当前最优位置相关.易见 $energy(P_i)\in[0,1]$.

定义 2.　设当前迭代次数为 $curGen$,$MAXGEN$ 为最大迭代次数,$eIni$ 与 $eFin$ 分别代表粒子能量的上界及下界,则对于给定的粒子 P_i,其能量阈值定义如下:

$$energyThreshold(P_i)=\left[\frac{(MAXGEN-curGen\times speed(P_i(curGen)))}{MAXGEN}\right]^e\times(eIni-eFin)+eFin,$$

其中,$speed(P(k)_i)=P_{\text{ibest}}(k)/P_{\text{ibest}}(k-1)$,$e$ 为预先指定的常量,用于控制粒子能量阈值的变化趋势.

粒子能量阈值与群体进化程度及粒子进化速度相关.可以看出,算法运行过程中粒子能量不断变化,当粒子能量小于它当前的能量阈值时,对其当前速度及位置执行变异操作如式(3)～(4)所示,以此引入新的信息增加粒子能量,扩大其能够到达的搜索范围.

$$V_i(k)=mutation(V_i(k)) \tag{3}$$

$$X_i(k)=mutation(X_i(k)) \tag{4}$$

以上模型在迭代过程中群体多样性会不断减少,全局搜索能力不断下降,影响群体的进化质量,如实验中 AHPSO-S 算法所示.由此,我们定义了粒子相似度及粒子相似度阈值,采用排序策略保持群体的多样性.粒子相似度用于度量两个粒子的相近程度,根据相邻两个粒子的个体最优位置的距离定义.

定义 3.　给定粒子 P_i,P_j,它们的相似度计算如下:

$$similarity(P_i,P_j)=\frac{\sum_{k=1}^{dim}same(P_{\text{ibest}}(k),P_{\text{jbest}}(k))}{dim},$$

其中,dim 表示待加工的作业数量.

定义 4.　设最大迭代次数 $MAXGEN$,粒子相似度阈值的取值范围为$[sIni,sFin]$,则当迭代次数为 $curGen$ 时粒子相似度阈值定义如下:

$$simiThreshold(curGen)=\left[\frac{(MAXGEN-curGen^*}{MAXGEN}\right]^s\times(sIni-sFin)+sFin,$$

其中 s 为一常量,用于控制粒子相似度阈值每次变化的幅度.

粒子相似度阈值用于设定当前群体中粒子之间距离的下界,它随迭代次数动态自适应变化.在算法运行的初始阶段,粒子相似度阈值取值较大,使得粒子在搜索空间中分布均匀,搜索范围扩大;随着群体的不断进化,相似度阈值不断减小,使得粒子之间能够逐渐聚合到当前的全局最优位置,加强搜索最优位置的邻域,进一步提高最优解的精度.为了保持群体的多样性,在进化过程中,根据适应度值对群体中的所有粒子进行排序,当两个相邻的粒子的相似度小于当前的粒子相似度阈值时,对较差粒子的历史最优解执行变异操作,如式(5)所示.

$$P_{\text{ibest}}(k+1)=mutation(P_{\text{ibest}}(k+1)) \tag{5}$$

通过变异操作能够在群体中重新引入新的有用信息,指导粒子搜索那些未曾搜索过的区域,进一步抑制算法的早熟收敛.

2.2　适应度计算

为了提高算法的速度,我们设计了一种基于遍历矩阵的快速计算粒子适应度的方法.对于给定的 n 个待加工作业,每个作业包含 m 道工序,我们将调度 $S=\langle s_1,s_2,\cdots,s_n\rangle$ 的加工时间矩阵 $\boldsymbol{T}$ 定义为

$$\boldsymbol{T}=\begin{bmatrix}t_{1,s_1} & t_{1,s_2} & \cdots & t_{1,s_n}\\ t_{2,s_1} & t_{2,s_2} & \cdots & t_{2,s_n}\\ \vdots & \vdots & \vdots & \vdots\\ t_{m,s_1} & t_{m,s_2} & \cdots & t_{m,s_n}\end{bmatrix},$$

其中,$s_i\in J$,t_{jk} 表示作业 j 在第 k 台处理机上的加工时间,$1\leq i\leq n$,$1\leq j\leq m$.

算法中粒子的适应度值由最小完工时间表示,根据以上定义,通过对问题数学模型的分析,给定的调度 S 对应的最小完工时间,可通过按照如下公式

遍历矩阵 $\boldsymbol{T}$ 求得：

$$t_{i,j}=\begin{cases}t_{1,1}, & i=1,j=1\\ t_{1,j-1}+t_{1j}, & i=1,j\neq 1\\ t_{i-1,1}+t_{i,1}, & i\neq 1,j=1\\ t_{i-1,j}+t_{i,j}, & t_{i-1,j}>t_{i,j-1}\\ t_{i,j-1}+t_{i,j}, & t_{i-1,j}<t_{i,j-1}\end{cases}.$$

遍历完成后，新计算出的 $t_{m,n}$ 的值就是调度 S 的最小完工时间.

2.3 算法描述及分析

在我们的 AHPSO 算法中，群体的初始化采用随机的方式，即在搜索空间中随机地产生粒子的初始位置和初始速度，将每个粒子的历史最优位置设置为其当前位置，并计算每个粒子当前位置所代表调度的最小完工时间，将其作为粒子的适应度值. 根据算法的进化模型，AHPSO 算法可描述如下.

算法. AHPSO.

```
begin
  initialize(pop);
  curGen := 0;
  while curGen<MAXGEN do
    for j := 1 to popNum do
     pop[j].v[curGen+1]:= pop[j].v[curGen]⊗
       gBest⊗pop[j].pBest;
     pop[j].x[curGen+1] := pop[j].x[curGen]⊗
       pop[j].v[curGen+1];
     pop[j].fitness := C_max(pop[j].current);
     if pop[j].fitness<C_max(pop[j].pBest) then
       pop[j].pBest := pop[j].current;
     endif
     if pop[j].fitness<C_max(gBest) then
            gBest:= pop[j].current; endif.
     c := j;
     while pop[c].fitness<pop[c-1]&&c!=1 do
          swap(pop[c],pop[c-1]);
          c := c-1;
     endwhile
    endfor
    for k := 1 to popNum do
      particleEnergy :=
            particleEnergyComputing(pop[k]);
      if particleEnergy<
        particleEnergyThreshold(pop[k],curGen) then
            increaseParticleEnergy(pop[k]);
      endif
      if k!=1 then
        similarity :=
            particleSimilarityComputing(pop[k]);
        if similarity<similarityThreshold(curGen) then
            pop[k].pBest := mutation(pop[k].pBest);
        endif
      endif
    endfor
    curGen++;
  endwhile
  return gBest;
end.
```

注：*MAXFGEN* 表示算法的最大迭代次数；*curGen* 及 *popNum* 分别为当前迭代次数和群体规模；*random* 表示[0,1]之间的随机数.

收敛通常是指一个系统或过程达到一个稳定状态，对基于群体的优化算法来说，算法的收敛可以根据群体的行为来定义[10]. 给定待求解的调度问题 P，其搜索空间 Ω，$gbest(t)\in\Omega$ 为算法在时间 t 或在群体的第 t 次进化中求得的最优位置. $gbest^*$ 为 Ω 中的一个固定位置，收敛的定义可记为

$$\lim_{t\to\infty} gbest(t)=gbest^*.$$

也就是说，如果由算法求得的 *gbest* 不在变化，那么就说处于收敛状态. 如果 *gbest* 为搜索空间的全局最佳位置，则算法获得了全局最优收敛，否则算法陷入局部最优位置.

定理 1. AHPSO 算法是收敛的.

证明. 将群体中的每个粒子的当前位置视为一个状态，则所有的粒子当前位置的集合可视为一种状态分布. 这种状态分布会随算法的运行而改变. 由于 AHPSO 算法的运行具有随机性，其基本操作只与当前状态有关，是无后效性的，因此可以把群体内的个体视为一个具有不同状态的随机变量的概率分布.

首先证明由式(1)～式(2)构成的迭代过程是收敛的. 设群体规模为 m，问题规模为 l 群体的状态记为 $(p_1,p_2,\cdots,p_m)$，最佳调度为 p^*. 所有的群体状态构成一个 $1\times N$ 的行向量，其中 $N=(m.l)!$，这个行向量的每个元素由排在一起的 m 个粒子的当前位置构成. 可表示为

$$\{(p_1^{(1)},p_2^{(1)},\cdots,p_m^{(1)}),(p_1^{(2)},p_2^{(2)},\cdots,p_m^{(2)}),\cdots,(p_1^{(N)},p_2^{(N)},\cdots,p_m^{(N)})\},$$

假设经过若干代进化后，达到种群最优状态 p^*，不妨设其排在状态行向量的第一个分两处，即

$$\{(p_1^{(*)},p_2^{(*)},\cdots,p_m^{(*)}),(p_1^{(2)},p_2^{(2)},\cdots,p_m^{(2)}),\cdots,(p_1^{(N)},p_2^{(N)},\cdots,p_m^{(N)})\},$$

根据粒子的速度及位置更新公式可知，此时，处在最优位置粒子的速度及历史最优位置均为 p^*，在下一次迭代中求得的粒子当前位置不变. 状态转移矩阵

可写为

$$\boldsymbol{C}=\begin{bmatrix}\mathbf{1} & \boldsymbol{\phi}\\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22}\end{bmatrix}$$

其中 $\boldsymbol{C}$ 为随机矩阵，其每一行至少有一个正的元素，$\boldsymbol{\phi}$ 为 $N-1$ 维行向量. 显然 $\boldsymbol{C}$ 为可约随机矩阵，且 $\boldsymbol{C}_{21}$，$\boldsymbol{C}_{22}$ 不是零矩阵，根据可约随机矩阵的性质有

$$\boldsymbol{C}^{\infty}=(C_1^{\infty},0,\cdots,0),$$

其中，$C_1^{\infty}>0$. 因为 $\boldsymbol{C}^{\infty}$ 可看作是一个状态分布，所以应有 $C_1^{\infty}=1$，于是 $\boldsymbol{C}^{\infty}=(1,0,\cdots,0)$，即算法收敛到了 p^*.

当算法中粒子能量小于其当前的能量阈值或粒子的相似度大于当前相似度阈值时，虽然引入了变异操作，但并没有改变当前已经得到的历史全局最优位置，而且在以后的进化中，只有当得到的位置由于此最优位置时才会被取代，即全局最优位置总是朝着更好的位置进化. 所以，根据定义 6，AHPSO 算法是收敛的. 证毕.

定理 2. AHPSO 算法求得的解是一个合法调度.

证明. 算法中使用的交叉及变异操作可参看文献[8]，每种操作都满足“alldifferent”约束，都是一个合法调度，即 AHPSO 算法的搜索空间是由所有合法调度构成的. 所以由 AHPSO 算法求得的解必然合法. 证毕.

定理 3. AHPSO 算法的空间复杂度为 $O((4n+2m+2)\cdot d)$，群体每次进化的最坏渐进时间复杂度为 $O(mnd^2)$. 其中 n 为群体规模，d 表示作业数量，m 为处理机个数.

证明. 在 AHPSO 算法运行过程中需要存储每个粒子的当前位置、个体最佳位置、上一次迭代的个体最佳位置及速度，令群体规模为 n，则它所消耗存储空间为 $O(4n\cdot d)$；在求解粒子适应度时还需要存储处理时间矩阵，所消耗的空间为 $O(2md)$；每次执行交叉及变异等操作时还需要一个存储新位置或速度的空间，此外还需要存储一个全局最优调度；所以 HPGA 算法的空间复杂度为 $O((4n+2m+2)\cdot d)$. 算法运行时，执行一次交叉操作消耗的时间最多为 $O(2d-1)$，变异操作最多为 $O(d)$，所以在 HPGA 算法的一次迭代过程中，由交叉或变异引起的最坏时间复杂度为 $O(2n(2d-1))$；计算个体能量时，需要访问每个粒子，由此引起的时间复杂度为 $O(2n\cdot d)$；计算相似度及相似度阈值所消耗的时间也为 $O(2n\cdot d)$，求解每个粒子适应度时需要求得及遍历与之相应的时间矩阵，其时间复杂度为 $O(2md)$. 由于在一次迭代中需要计算所有粒子的适应度，那么需要消耗的时间就变为 $O(2mnd)$，并且最坏情况下，在一次迭代中每个粒子都需要更新其个体最优位置及当前位置和速度时，那么此时所消耗的时间就变为 $O(4mnd2)$. 当问题规模逐渐增大即 $m\cdot n$的值趋于无穷大时，$4mnd^2\geq 8nd\geq 2n$，所以 HPGA 算法的最坏时间渐进复杂度为 $O(mnd^2)$.

证毕.

2.4　算法改进

实验中发现，进化后期算法收敛速度明显下降，当接近最优解时，易于停止进化. 为此 AHPSO 算法中引入一种基于邻域的贪心随机搜索策略在每次迭代过程中更新粒子的个体最优解，进一步提高解的质量，此时将算法记作 G-AHPSO.

定义 5. 给定一个含有 n 个作业的调度问题，它的任意一个有效调度 $\pi=\{\pi_1,\cdots,\pi_n\}$ 可看作是 n 维空间中的一个点. 通过随机选择一个作业 π_i，$1\leq i\leq n$，将其插入到任意其它位置，可求得一个新的有效调度，即 n 维空间中一个新的点. 所有以此方式求得的点就构成了作业 π_i 的邻域.

根据以上定义，我们提出一种基于邻域的贪心随机搜索策略，可描述如下.

```
procedure. NegihborhoodGreedy_Insersion(π).
begin
  improve := true;
  while (improve=true) do
    improve := false;
    i := random(0,dimension−1);
    remove the job π_i from π;
    π′:= best schedule obtained by inserting π_i in any
         positions of π;
    if C_max(π′)<C_max(π) then
      π := π′;
      improve := true;
    endif
  endwhile
  return π;
end.
```

3　实验结果

文献[8]对各种基本的遗传操作及它们对 GA 算法求解 FSP 问题的性能的影响进行了分析. 实验中本文选择第二种形式的两点交叉操作，并分别测试了 6 种变异操作对算法的影响. 这 6 种变异操作分别是近邻变异(M1)、随机交换变异(M2)、移位变

异(M3)、置换变异(M4)、逆转变异(M5)、逆转/置换变异(M6).此外,还测试了粒子能量及粒子相似度策略对算法性能的影响.最后在不同规模的 Taillard 数据集[11]上对 AHPSO 及 G-AHPSO 算法进行了测试,并与最近提出的求解调度问题的 GA 算法[8]、SPSOA 算法[7]进行了比较.为了方便,实验中 AHPSO 和 G-AHPSO 算法的参数设置相同,$MAXGEN=1000$, $popNum=60$, $s=1.40$, $e=1.35$, $eIni=0.45$, $eFin=0.10$, $sIni=0.85$, $sFin=0.05$.

为说明算法每次求得的最优解与已知最优解的差距,我们定义了算法所求得的最优解的平均偏离度(Average Relative Deviation, ARD)即算法每次运行得到的最优解与已知最优解的差的均值.计算如下:

$$ARD=\sum_{i=1}^{T}(Sol_i-Optimal)/T,$$

其中 T 为针对某个问题算法的运行次数,它是算法求得的最优解偏离已知最优解平均程度的指标,能够反映出算法的平均求解质量.本文实验中每个测试问题上各算法运行次数都为 10,即 $T=10$.

3.1 粒子能量及粒子相似度策略对算法性能的影响

为了测试它们在 AHPSO 算法中的作用,我们比较了采用移位变异操作的 AHPSO 算法、AHSPO-S 算法及 AHPSO 算法在不同规模问题上的求解结果.AHPSO-S-A 算法是去除粒子能量及粒子相似度策略即迭代模型只由式(1)~(2)构成的算法;AHPSO-S 算法是去除粒子相似度策略即迭代模型由式(1)~(4)构成的算法.表 1 为在每个问题上各算法 10 次运行中求得的最好解、最优解均值、最差解及平均偏离度的比较结果.各算法在不同规模问题上的群体平均适应度进化曲线如图 1 所示.

表 1 AHPSO,AHPSO-S 和 AHPSO-S-A 在不同规模问题上的测试结果比较、括号内的值为算法在此问题上运行 10 次的平均偏离度

问题	测试结果		
	AHPSO-S-A	AHPSO-S	AHPSO
ta005	1250/1254.3/1277 (19.3)	1235/1242.0/1250 (7.0)	1235/1242.7/1250 (7.7)
ta010	1120/1139.2/1161 (31.2)	1108/1108.0/1108 (0.0)	1108/1108.0/1108 (0.0)
ta020	1614/1632.7/1654 (41.7)	1599/1614.2/1631 (23.2)	1598/1611.4/1618 (20.4)
ta030	2212/2240.3/2269 (62.3)	2194/2204.4/2221 (26.4)	2178/2190.3/2199 (12.3)
ta050	3190/3223.9/3245 (158.9)	3131/3161.1/3204 (96.1)	3112/3156.0/3204 (91.0)
ta060	3982/4019.9/4075 (263.9)	3884/3917.6/3969 (161.6)	3843/3872.5/3899 (116.5)
ta070	5342/5394.8/5429 (72.8)	5328/5340.2/5346 (18.2)	5328/5336.6/5346 (14.6)
ta080	5986/6047.4/6087 (202.4)	5896/5902.4/5904 (57.4)	5881/5900.8/5903 (55.8)

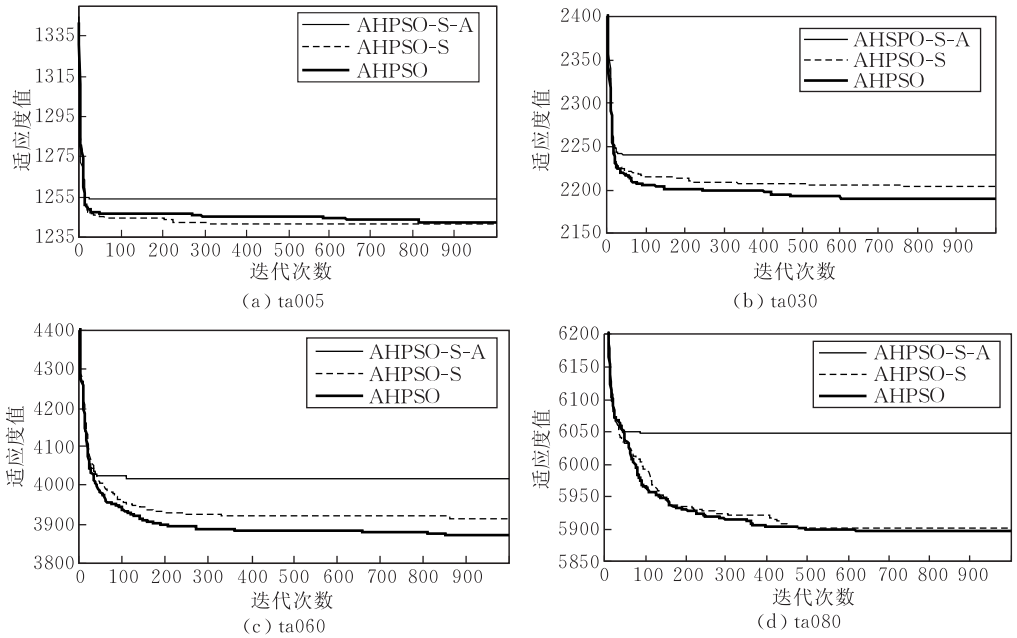

图 1 不同规模问题上 AHPSO、AHPSO-S 和 AHPSO-S-A 算法的群体平均进化曲线

可以看出 AHPSO 算法的平均求解质量最佳,求得的最优解及平均偏离度都是最小的.AHPSO-S 的求解质量也明显高于 AHPSO-S-A 算法.也就是说粒子能量及粒子相似度策略能够极大地提高算法

的性能.从图 1 中明显看出 AHPSO-S-A 算法虽然收敛速度快,但容易陷入局部最优解,其求解质量最差.此外,对于小规模问题由于其解空间小,粒子能够搜索到的区域几乎涵盖整个解空间,此时,粒子相似度策略对算法性能影响不大,如图 1(a)所示.随着问题规模增加,其求解空间也变得相对复杂,此时排序策略能够指导粒子朝着最有希望获得更好解的区域搜索,进一步提高算法的性能,如图 1(b)～(d)所示.

3.2　局部搜索策略对算法性能的影响

为了测试局部搜索策略对算法性能的影响,我们在每个测试集上采用不同变异操作的 AHPSO 及 G-AHPSO 算法分别运行 10 次,所求得的最优解、最优解均值及最差解,如表 2 和表 3 所示.

表 2　采用不同变异操作的 AHPSO 算法的测试结果

问题	测试结果					
	M1	M2	M3	M4	M5	M6
ta005	1243/1252.6/1294	1235/1246.3/1250	1235/1242.7/1250	1244/1247.0/1250	1235/1242.8/1250	**1235/1240.4/1250**
ta010	1108/1125.3/1138	1108/1111.8/1127	**1108/1108.0/1108**	1108/1109.2/1120	1108/1108.9/1111	1108/1110.1/1120
ta020	1614/1628.8/1654	1591/1610.9/1620	1608/1611.4/1618	1591/1609.9/1629	1608/1611.8/1615	**1591/1606.1/1618**
ta030	2210/2238.0/2273	2185/2198.0/2227	**2178/2190.3/2199**	2179/2184.6/2196	2186/2201.9/2220	2183/2195.6/2207
ta050	3204/3222.3/3286	3131/3162.6/3191	**3112/3156.0/3204**	3131/3156.9/3216	3131/3163.8/3213	3152/3176.2/3195
ta060	3946/4000.8/4076	3871/3906.8/3965	**3843/3872.5/3899**	3859/3892.1/3908	3884/3924.2/3985	3868/3913.9/3944
ta070	5342/5379.5/5422	5342/5343.2/5346	**5328/5340.5/5346**	5342/5352.0/5386	5342/5355.8/5386	5342/5343.0/5346
ta080	5949/6036.4/6087	5903/5904.3/5916	**5881/5900.8/5903**	5903/5914.8/6009	5903/5967.8/6069	5903/5906.6/5924

表 3　采用不同变异操作的 G-AHPSO 算法的测试结果

问题	测试结果					
	M1	M2	M3	M4	M5	M6
ta005	1235/1241.5/1250	1235/1235.8/1243	**1235/1235.0/1235**	1235/1236.8/1244	**1235/1235.0/1235**	**1235/1235.0/1235**
ta010	1108/1111.8/1127	**1108/1108.0/1108**	**1108/1108.0/1108**	**1108/1108.0/1108**	**1108/1108.0/1108**	**1108/1108.0/1108**
ta020	1604/1610.3/1614	1591/1603.6/1613	1591/1607.1/1617	1591/1606.3/1608	1591/1599.7/1613	**1591/1598.7/1608**
ta030	2178/2189.9/2202	2178/2181.3/2189	2179/2182.8/2195	**2178/2181.0/2185**	2178/2181.2/2188	2179/2181.0/2183
ta050	3131/3139.0/3162	3091/3119.8/3132	**3091/3110.3/3131**	3095/3118.9/3132	3092/3128.1/3146	3091/3118.5/3139
ta060	3849/3870.0/3899	3800/3822.5/3855	3806/3839.8/3899	3797/3829.2/3861	**3790/3825.7/3855**	3802/3829.1/3864
ta070	5324/5328.5/5332	5324/5331.0/5342	5328/5331.6/5342	5328/5329.6/5342	5324/5329.4/5342	**5322/5331.6/5342**
ta080	5881/5900.8/5903	5881/5896.4/5903	5881/5900.8/5903	**5856/5887.3/5903**	5856/5891.7/5903	5879/5898.4/5903

从表中可以看出,无论是从所得到的最优解、最优解均值还是最差解方面,使用变异操作 M3 时 AHPSO 算法的性能最佳,在 10 次运行中除问题 ta020 外,求得的最小完工时间都优于使用其它几种变异操作时算法求得的解.对于 G-AHPSO 算法来说,除变异操作 *M*1 外,使用其它几种变异操作对算法的性能影响不是很大,这主要是由于使用这几种变异操作时,算法的全局搜索能力相差不大,而在算法运行后期,解的质量的精化又主要依赖于贪心随机搜索策略.从表 1、表 2 的对比中可以看出,无论使用哪种变异操作,G-AHPSO 算法求得的最优解、最优解均值和最差解都要明显好于 AHPSO 算法求得的解.

此外,我们还比较了这两种算法的收敛速度,如图 2,它显示了对于不同规模的问题,采用不同变异操作时最小完工时间随迭代次数的进化曲线.其中虚线表示 AHPSO 算法的收敛曲线,实线是 G-AHPSO 算法的收敛曲线.显然,G-AHPSO 算法随迭代次数收敛得快些,而且收敛点的值也小于 AHPSO 算法.对于不同的问题,变异操作对算法的收敛速度的影响也稍有不同.

AHPSO 算法与 G-AHPSO 算法在各问题上所求得解的平均偏离度,如表 4 所示,括号内为 G-AHPSO 算法的平均偏离度.

表 4　AHPSO 与 GAHPSO 算法平均偏离度的比较

问题	测试结果					
	M1	M2	M3	M4	M5	M6
ta005	17.6(6.5)	11.3(0.8)	7.7(0.0)	12.0(1.8)	7.8(0.0)	**5.4(0.0)**
ta010	17.3(3.8)	3.8(0.0)	**0.0(0.0)**	1.2(0.0)	0.9(0.0)	2.1(0.0)
ta020	37.8(19.3)	19.9(12.6)	20.4(16.1)	18.9(15.3)	20.8(8.7)	**15.1(7.7)**
ta030	60.0(11.9)	20.0(3.3)	12.3(4.8)	**6.6(3.0)**	23.9(3.2)	17.6(3.0)
ta050	157.3(74.0)	97.6(54.8)	**91.0(45.3)**	91.9(53.9)	98.8(63.1)	111.2(53.5)
ta060	244.8(114.0)	150.8(66.5)	**116.5(83.8)**	136.1(73.2)	168.2(69.7)	157.9(73.1)
ta070	57.5(6.5)	21.2(9.0)	**18.5(9.2)**	30.0(7.6)	33.8(7.4)	21.0(9.6)
ta080	191.4(55.8)	59.3(51.4)	**55.8(55.8)**	69.8(42.3)	122.8(46.7)	61.6(53.4)

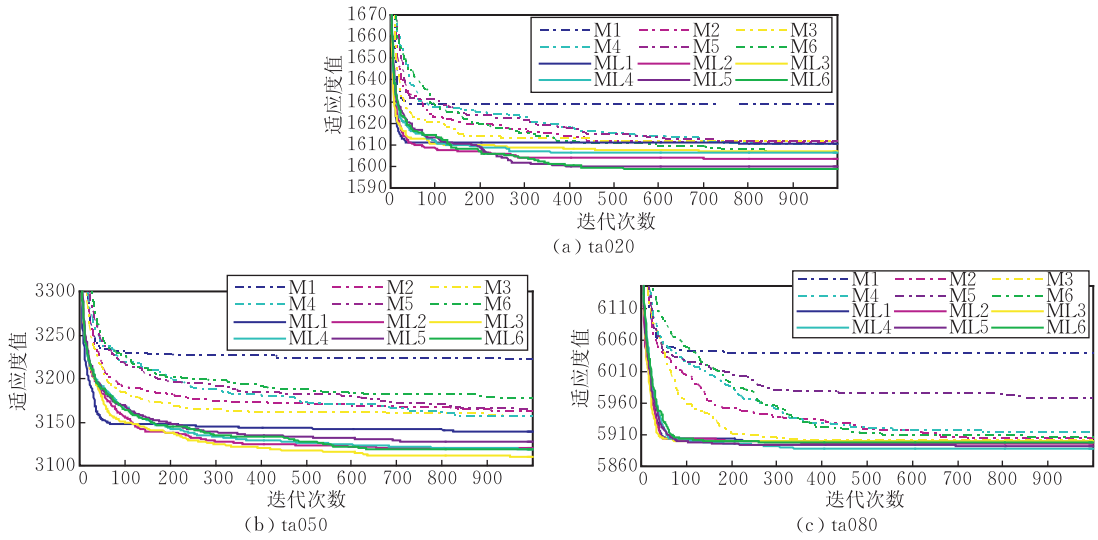

图 2 不同规模问题上采用各种变异操作时 AHPSO 和 GAHPSO 算法的群体适应度进化曲线

可以看出,无论是使用哪种变异操作 G-AHPSO 算法求得解的平均偏离度都明显小于 AHPSO 算法求解的平均偏离度. 也就是说,G-AHPSO 算法每次求得高质量解的概率远远大于 AHPSO 算法.

从上面的实验分析对比中可以得出,无论是从求得的解质量、收敛速度还是平均偏离度方面,G-AHPSO 算法都明显优于 AHPSO 算法. 但是,由于 G-AHPSO 算法在群体进化中,每个粒子都要执行贪心随机搜索过程,所以每次群体进化所消耗的时间也高于 AHPSO 算法. 在所有测试问题上 AHPSO 算法与 G-AHPSO 算法进化一次所消耗的时间对比如图 3 所示.

此外,由于贪心随机搜索过程主要依赖于插入操作求解邻域内的点,所以在计算这些点的适应度时可以通过使用文献[12]中基于插入的加速算法可以进一步提高 G-AHPSO 算法的速度,但仍会略慢于AHPSO 算法. 在实际应用中是否采用贪心搜索策略,需要在求解质量与求解速度之间进行权衡.

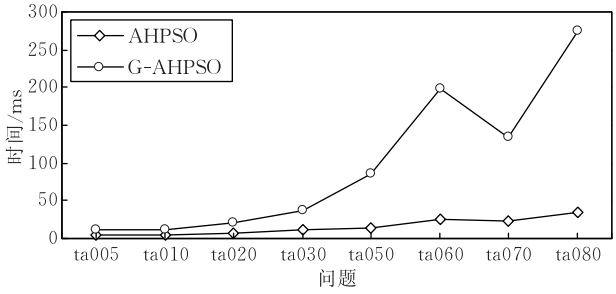

图 3 采用变异操作 M3 时 AHPSO 和 G-AHPSO 算法求解各问题的时间对比

3.3 与其他算法的比较

最后,为了进一步说明本文提出算法的有效性,将它们与最近提出的 GA 算法[8] 及 SPSOA 算法[7] 进行了比较,所得结果如表 5 所示. 表中都是使用各变异操作每个算法运行 10 次得到的最好结果,可以看出在所有的测试集上无论 G-AHPSO 算法还是 AHPSO 算法所求得的最优解、最优解均值及最差解都明显小于其他两种算法,而且 G-AHPSO 算法能够求得问题 ta070 的最优解 5322,也就是说在此问题上 G-AHPSO 具有全局收敛性.

表 5 算法求解结果比较

问题	规模	最优解	求解结果			
			GA	SPSOA	AHPSO	G-AHPSO
ta005	20×5	1235	1304/1347.0/1387	1244/1250.4/1254	**1235/1240.4/1250**	**1235/1235.0/1235**
ta010	20×5	1108	1188/1252.4/1303	1108/1126.0/1160	**1108/1108.0/1108**	**1108/1108.0/1108**
ta020	20×10	1591	1702/1790.7/1852	1600/1635.5/1674	**1591/1606.1/1618**	**1591/1598.7/1608**
ta030	20×20	2178	2368/2453.9/2517	2194/2243.0/2299	**2178/2190.3/2199**	**2178/2181.0/2185**
ta050	50×10	3065	3384/3512.3/3557	3173/3220.4/3260	**3112/3156.0/3204**	**3091/3110.3/3131**
ta060	50×20	----	4338/4458.2/4524	3917/3982.5/4038	**3843/3872.5/3899**	**3790/3825.7/3855**
ta070	100×5	5322	5409/5611.6/5688	5342/5362.8/5389	**5328/5340.5/5346**	**5322/5331.6/5342**
ta080	100×10	5845	6302/6395.0/6493	5903/5959.1/6051	**5881/5900.8/5903**	**5856/5887.3/5903**

4 结　语

本文工作主要体现在以下几个方面：(1) 提出了一种求解流水调度问题的自适应混合粒子群算法，将遗传操作结合到算法中；(2) 定义了粒子能量及随进化程度自适应调整的粒子能量阈值，采用变异操作保持粒子的搜索能力；(3) 引入了粒子相似度及相似度阈值，并采用排序策略保持群体的多样性，抑制算法的早熟收敛；(4) 通过使用遍历矩阵方式快速计算一个调度的最小完工时间；(5) 证明了算法的收敛性，分析了算法空间复杂度及迭代一次的渐进时间复杂度. 最后定义了衡量算法性能指标-平均偏离度，将该算法在 8 个不同规模的问题上进行了测试，并与当前国际文献中最近提出的两种著名算法进行了比较，结果表明无论是求得的解得质量还是算法的稳定性均明显好于这两种算法，加入随机贪心搜索策略后效果更佳明显. 下一步工作主要集中在测试其他交叉策略对算法的影响，找出交叉操作与变异操作的最佳组合，以及使用本文提出的算法解决其他组合优化问题.

参 考 文 献

[1] Garey M, Johnson D, Sethy R. The complexity of flow shop and job shop scheduling. Mathematics of Operations Research, 1976, 1(2): 117-129

[2] Valente J M S, Alves R A F S. An exact approach to early/tardy scheduling with release dates. Computers & Operations Research, 2005, 32(11): 2905-2917

[3] Gangadharan R, Rajendran C. Heuristic algorithms for scheduling in no-wait flowshop. International Journal of Production Economics, 1993, 32(3): 285-90

[4] Aldowaisan T, Allahverdi A. New heuristics for no-wait flowshops to minimize makespan. Computers and Operations Research, 2003, 30(12): 19-31

[5] Yang Qing-Yun, Sun Ji-Gui, Zhang Ju-Yang et al. A hybrid discrete particle swarm algorithm for open-shop problems// Proceedings of the 6th International Conference on Simulated Evolution and Learning (SEAL 2006). Hefei, China, 2006: 158-165

[6] Rameshkumar K, Suresh R K, Mohanasundaram K M. Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makspan//Proceedings of the ICNC 2005, Changsha, China, 3612, 2005: 572-581

[7] Lian Zhi-Gang, Gu Xing-Sheng, Jiao Bin. A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. Applied Mathematics and Computation, 2006, 175(1): 773-785

[8] Nearchou A C. The effect of various operators on the genetic search for large scheduling problems. International Journal Production Economics, 2004, 88(12): 191-203

[9] Lauriere J-L. A language and a program for stating and solving combinatorial problems. Artificial Intelligence, 1978, 10(1): 29-127

[10] van den Bergh F. An analysis of particle swarm optimizers [Ph. D. dissertation]. Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002

[11] Taillard E. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research, 1990, 47(1): 65-74

[12] Pan Quan-Ke, Tasgetiren M F, Liang Yun-Chia. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Computers & Operations Research, 2008, 35(9): 2807-2839

ZHANG Chang-Sheng, born in 1980, Ph. D.. His current research interests include intelligent information processing and constraint programming.

SUN Ji-Gui, born in 1962, Ph. D., professor, Ph. D. supervisor. His research interests include constraint programming and automatic reasoning.

OUYANG Dan-Tong, born in 1968, Ph. D., professor, Ph. D. supervisor. His research interests include automatic reasoning and model based Diagnosis

ZHANG Yong-Gang, born in 1975, Ph. D.. His current research interests include intelligent information processing and constraint programming.

Background

The scheduling problems is a kind of important optimization problems in real life which is NP-hard and difficult to find a satisfying solution within a limited time. How to get the best or a satisfying solution quickly has great significance for improving productivity and the development of society. The traditional optimization algorithms have many advanta-

ges such as good computing efficiency, strong soundness and mature property, but they all have insurmountable limitation that it is difficult or impossible to find the exactly or even approximately best solution for a complex system. The presentation of evolutionary algorithm gives a new idea for solving complex optimization problems. It has been adopted by many research fields because of its intelligence, universality, stability, essential parallelism and global search ability. Many researchers in our country have put forward lots of optimization algorithms in which the particle swarm optimization algorithm is newer and preferable. It has been applied to many project practice problems and can get very good optimization effects. For many years the main focus of research was on the application of single metaheuristics to given problems. In recent years, it has become evident that the concentration on a sole metaheuristic is rather restrictive. A skilled combination of concepts of different metaheuristics, called hybrid metaheuristic, can provide a more efficient behavior and a higher flexibility when dealing with real-world and large-scale problems. In this work, the authors combined the generic operations with the particle swarm optimization algorithm and propose algorithm for the flow shop scheduling problem with single objective. Based on the analysis for the algorithm, its effectivenesses are validated by the comparisons with other existing algorithms on benchmarks with different scale and difficulties. Hybrid metaheuristic is a new research area and currently enjoys an increasing interest in the optimization community since the special issue "Hybrid Metaheuristics" of International Journal of Mathematical Modelling and Algorithms was published in 2005. The design and implementation of hybrid metaheuristics rise problems going beyond questions about the design of a single metaheuristic. Choice and tuning of parameters is for example enlarged by the problem of how to achieve a proper interaction of different algorithm components. Interaction can take place at low-level, using functions from different metaheuristics, but also at high-level, e. g. , using a portfolio of metaheuristics for automated hybridization. However, the field of hybrid metaheuristics is still in its early days. Many approaches are pre-mature and a substantial amount of further research is necessary in order to develop clearly structured hybrid metaheuristics that can be generally used for optimization. So, this work in this paper will enrich and push forward the studies of the related areas in both theoretical and technological aspects.