文章编号: 1001-0920(2018)08-1363-10 DOI: 10.13195/j.kzyjc.2017.0524

差异容量平行批机器环境下基于弱选择约束的调度算法

贾兆红[1,2†], 杨 洋[2], 张以文[2]

(1. 安徽大学 计算智能与信号处理教育部重点实验室，合肥 230039；
2. 安徽大学 计算机科学与技术学院, 合肥 230601)

摘 要: 研究动态到达的差异工件在容量不同的平行批处理机环境下,最小化制造跨度的调度问题,并提出一种有效的元启发式算法. 给出一个下界以评价算法的性能,针对所构建批的第1个工件的选择提出弱约束标准及两个基于弱约束的首工件选择策略,并引入到蚁群优化算法. 最后通过仿真实验将所提出的改进蚁群算法与已有算法和使用传统选择策略的蚁群算法进行比较,实验结果表明,在建批过程中使用首工件弱约束策略和弱约束下工件尺寸大高概率选择策略是有效的,所提算法的搜索性能较其他算法具有明显优势.

关键词: 批调度；平行机；差异机器容量；弱约束；蚁群算法

中图分类号: TP301 **文献标志码:** A

Weak-restriction based algorithm for scheduling on parallel batch machines with arbitrary capacities

JIA Zhao-hong[1,2†], *YANG Yang*[2], *ZHANG Yi-wen*[2]

(1. Key Lab of Intelligent Computing and Signal Processing of Ministry of Education，Anhui University，Hefei 230039，China；2. School of Computer Science and Technology，Anhui University，Hefei 230601，China)

Abstract: To minimize the makespan for scheduling the jobs with non-identical sizes and dynamic arrivals on the parallel batch process machines with non-identical capacities, an effective meta-heuristic is proposed. A lower bound is presented to measure the performance of algorithm. Based on the weak restriction for the first jobs of the batches under construction, two selection strategies are proposed and used to improve the ant colony optimization(ACO) algorithm. The proposed algorithm is compared with several up-to-date algorithms and the ACO algorithm using usual selection method by simulations. The results demonstrate the effectiveness of selecting the first jobs used in the proposed ACO algorithm, the weak restriction where the jobs with larger sizes have larger probability to be selected, and the proposed algorithm has obvious superiority in search performance than the others.

Keywords: batch scheduling; parallel machines; arbitrary machine capacities; weak restriction; ant colony optimization

0 引 言

批调度问题是一类复杂的离散优化问题. 不同于经典调度问题,在批调度问题中,多个工件可以同时在一个批处理机中加工,该问题与实际生产有紧密联系,如半导体元器件测试、工业制造、货物运输等. 因此,研究批调度问题具有重要的现实意义. 求解批调度问题时,首先对工件进行分批,然后将工件以批的形式分配到机器上加工. 工件分批时,需要保证批中所有工件的尺寸之和不能超过加工该批的机器容量. 工件分批后,批的加工时间取决于该批中工件的加工时间,批的开始加工时间受到该批前面相邻的批的完工时间和该批中工件的最迟到达时间影响.

近年来,单机批调度问题已经得到较为广泛的研究. Ikura等[1]首先研究了单机批调度问题,并提出了一种动态规划方法对工件尺寸相同的单机批调度问题进行求解;Uzsoy[2]分别研究了极小化最大制造跨度和极小化总完工时间的差异工件单机批调度问题,证明了这两个问题均为强NP难问题,并提出了若干启发式算法,实验结果表明,在极小化最大制造跨度时,FFLPT(First fit longest processing time)规则的

收稿日期: 2017-04-27；**修回日期:** 2017-07-13.

基金项目: 国家自然科学基金项目(71601001,71671168)；教育部青年基金项目(15YJC630041)；安徽省自然科学基金项目(1608085MG154)；安徽省教育厅自然科学基金项目(KJ2015A062).

责任编委: 魏秀琨.

作者简介: 贾兆红(1976−), 女, 副教授, 博士, 从事计算智能及其应用等研究；杨洋(1992−), 男, 硕士生, 从事商务智能的研究.

†通讯作者. E-mail: jiazhaohong001@163.com

性能相对更优; Dupont等[3]研究了差异工件单机批调度问题,提出了启发式算法BFLPT(Best fit longest processing time); Dupont等[4]提出了分支限定法求解目标为极小化总完工时间的差异工件单机批调度问题.

随着智能优化算法的兴起,基于智能优化算法的批调度研究越来越多. Melouk等[5]采用模拟退火(SA)算法求解了差异工件单机批调度问题; Kashan等[6]采用两种基于不同编码规则的遗传算法(GA)来求解差异工件单机批调度问题; Chou等[7]在GA中结合FFLPT启发式规则,并采用这种方法求解工件动态到达时间下单机批调度问题;王栓狮等[8]采用两种不同信息素表达的蚁群优化(ACO)算法求解工件含有不同到达时间且差异尺寸的单机批调度问题;许瑞等[9]提出了采用两种不同编码方式的ACO算法对该问题进行求解,实验表明,直接分批编码方式更具有优势.

上述研究均针对单机批调度问题,然而现实生产环境中,往往同时采用多台平行机. Xu等[10]提出了一种随机键遗传算法(RKGA)来求解在平行机环境下的批调度问题,与商业软件CPLEX的比较实验表明,在大规模问题上,RKGA可得到更好的结果; Cheng等[11]提出了一种改进的蚁群算法求解平行机批调度问题; Wang等[12]提出了基于启发式的GA和SA来求解差异容量平行机批调度问题,并通过实验表明了算法的有效性; Chen等[13]使用GA和ACO算法对工件具有不同到达时间的平行机批调度问题进行了求解,结果表明,当工件数增加时,ACO算法相对于GA更具有优势; Damodaran等[14]使用粒子群优化(PSO)算法对差异工件平行机批调度问题进行求解,并与RKGA进行比较,实验表明,PSO算法得到的结果更好; Zhou等[15]使用离散差分进化(DDE)算法来求解容量不同的平行机下差异工件批调度问题,并与PSO算法进行了比较; Jia等[16]使用ACO算法求解容量不同的平行机环境下差异工件动态到达的批调度问题,并通过实验验证了算法的有效性.

本文研究的是容量不同的平行批处理机环境下,差异工件动态到达的制造跨度最小化问题. 实际上,考虑到达时间、工件尺寸和机器尺寸均不同的问题较工件尺寸相同、到达时间相同或机器尺寸相同的问题更复杂. 现阶段大多数针对机器容量不同的平行机批调度问题的研究是基于单个工件尺寸小于最小机器容量的假设. 当工件尺寸和机器尺寸均不同,且某些大尺寸工件不能在小容量机器上加工时,问题的求解难度必定增加,而目前仅有文献[14,16]涉及了工件带有机器容量约束的问题. 在文献[14]中,所有工件均假设在0时刻到达,而本文考虑工件动态到达的约束,因而问题更加复杂.

本文通过分析批中第一个工件对正在构建的批存在较大影响,从而对第一个工件的选择进行约束,并结合这种思想提出基于ACO的改进算法. 本文首先给出该问题下界的求解方法,提出建批过程中首个工件弱约束策略,并在此基础上提出首工件选择策略. 基于弱约束策略和不同的工件选择策略,提出两种蚁群算法对问题进行求解,分别是弱约束下批中首工件工件尺寸大高概率选择算法(ACOL)和弱约束下批中首工件均等概率选择算法(ACOU). 最后通过仿真实验,将本文中所提出的算法与以往文献中已有算法进行对比以验证算法的性能.

1 问题描述

本文研究在容量不同的平行批处理机环境下,调度动态到达且带有任意尺寸的工件以最小化制造跨度的问题. 参照三参数表示法[16],该问题可以表示为$P_m|p\text{-batch}, r_j, p_j, s_j, S_i|C_{\max}$,包含如下假设.

1) 工件集合为$J = \{J_1, J_2, \cdots, J_n\}$,任一工件$J_j$的加工时间为$p_j$,尺寸为$s_j$,到达时间为$r_j$,每个工件只能放入一个批中,且只能放入一个机器中加工.

2) 批处理机集合为$M = \{M_1, M_2, \cdots, M_m\}$,机器$M_i$的容量为$S_i$(不妨令$S_1 \leqslant S_2 \leqslant \cdots \leqslant S_m$),$M_i$的完工时间$C_i$为该机器上所有批的最大完工时间. 在机器集$M$中有$h$种不同容量的机器,第$k$种容量的机器容量定义为$S^k$,不妨令$S^1 < S^2 < \cdots < S^h$, M^k表示机器容量为S^k的机器集合.

3) 工件分组形成的批集合为B,其中M_i上的第b个批记为B_{bi},批B_{bi}的到达时间R_{bi}等于批中工件的最大到达时间,即$R_{bi} = \max\{r_j | J_j \in B_{bi}\}$;批$B_{bi}$的加工时间$P_{bi}$等于批中工件的最大加工时间,即$P_{bi} = \max\{p_j | J_j \in B_{bi}\}$;批尺寸为批中所有工件的尺寸之和,即$\mathrm{BS}_{bi} = \sum_{J_j \in B_{bi}} \{s_j\}$. 批尺寸不得大于批所在机器的容量,即$\mathrm{BS}_{bi} \leqslant S_i$. 批一旦加工则不可中断.

4) 批B_{bi}调度到机器后,则可定义其开始加工时间ST_{bi}和完工时间C_{bi},其中ST_{bi}为批到达时间R_{bi}和同一个机器前一个批的完工时间$C_{(b-1)i}$的最大值,即$\mathrm{ST}_{bi} = \max\{R_{bi}, C_{(b-1)i}\}$,而$C_{bi} = \mathrm{ST}_{bi} + P_{bi}$.

5) 若机器M_i上有批序列$B_{1i}, B_{2i}, \cdots, B_{qi}$,则该机器的完工时间为序列中最后一个批的完工时间,即$C_i = C_{qi}$. 若机器上没有批加工,则该机器的完工时

间为0.

6) 优化目标为$C_{\max}$,即所有机器中最大完工时间$C_{\max}=\max\limits_{M_i\in M}\{C_i\}$.

单机批调度问题已被证明为NP难问题,而本文研究的问题更加复杂,所以该问题也是NP难的. 为了客观评价算法的性能,已有学者提出了使用松弛工件尺寸约束的方法得到单机批调度问题制造跨度的下界. Jia等[16]给出了本文所研究问题的一个下界$\mathrm{LB}_1=\max\Big\{\max\limits_{J_j\in J}\{r_j+p_j\},\min\limits_{J_j\in J}\{r_j\}+\Big\lceil\sum\limits_{i=1}^{z}P_i/m\Big\rceil\Big\}$,并证明了下界的正确性. 然而,当问题处于工件数随着到达时间的增加而增加的情形时,LB_1是较为松弛的下界,于是本文提出一个新的下界算法LB. 为了描述方便,在算法LB中引入一个变量w来记录工件集合J中工件到达时间不同值的数量.

算法LB.

Step 1: $\mathrm{LB}=\max\limits_{J_j\in J}\{r_j+p_j\}$.

Step 2: 令$v=1,\mathrm{JB}_v=J,\mathrm{JB}'_v=\varnothing$.

Step 3: 如果$v>w$,则输出LB,结束;否则,找出JB_v中具有最小到达时间$r=\min\limits_{J_j\in\mathrm{JB}_v}\{r_j\}$的工件子集$\mathrm{JB}'_v=\{J_j|J_j\in\mathrm{JB}_v\text{ and }r_j=r\}$.

Step 3.1: 将JB_v中的所有工件尺寸单位化,即任一工件J_j单位化为s_j个加工时间为p_j且到达时间为0的单位工件,所得单位工件集合为JB_v^U.

Step 3.2: 将JB_v^U中的工件按加工时间非递增排序.

Step 3.3: 将JB_v^U中每S_m个工件组成一个批,直到所有工件分组完成,这样得到z_v个批,其中z_v个批中最后一个批B_{z_v}的尺寸可能小于S_m,$z_v=\Big\lceil\sum\limits_{j_1=1}^{|\mathrm{JB}_v^U|}s_{j_1}/S_m\Big\rceil$,每个批的加工时间为$P_{i_1}$.

Step 3.4: 如果$\mathrm{LB}<r+\Big\lceil\sum\limits_{i_1=1}^{z_v}P_{i_1}/m\Big\rceil$,则$\mathrm{LB}=r+\Big\lceil\sum\limits_{i_1=1}^{z_v}P_{i_1}/m\Big\rceil$.

Step 3.5: $v{+}{+},\mathrm{JB}_v=\mathrm{JB}_{(v-1)}-\mathrm{JB}'_{(v-1)}$,转入Step 3.

算法LB所得的下界可以表示为

$$\mathrm{LB}=\max\Big\{\max_{J_j\in J}\{r_j+p_j\},\ \max_{v=1}^{w}\Big\{\min_{J_j\in\mathrm{JB}_v}\{r_j\}+\Big\lceil\Big(\sum_{i_1=1}^{z_v}P_{i_1}\Big)\Big/m\Big\rceil\Big\}\Big\}.\tag{1}$$

其中:w为工件集J中不同工件到达时间的数量,$\mathrm{JB}_1=\{J_j|J_j\in J\},\mathrm{JB}_2=\{J_j|J_j\in J\text{ and }r_j>\min\limits_{J_j\in\mathrm{JB}_1}\{r_j\}\},\cdots,\mathrm{JB}_w=\{J_j|J_j\in J\text{ and }r_j>\min\limits_{J_j\in\mathrm{JB}_{(w-1)}}\{r_j\}\}$. $z_1,z_2,\cdots,z_w$分别为工件集$\mathrm{JB}_1,\mathrm{JB}_2,\cdots,\mathrm{JB}_w$中工件的尺寸单位化后以$S_m$为单位分组后所得批的数量.

定理1　通过算法LB可以得到所求问题的一个有效下界.

证明　所研究问题的目标下界显然不会超过$\max\limits_{J_j\in J}\{r_j+p_j\}$. 文献[16]中已经证明了$\min\limits_{J_j\in J}\{r_j\}+\Big\lceil\sum\limits_{i=1}^{z}P_i/m\Big\rceil$为本文问题的合理下界. 由于$\mathrm{JB}_1=J$,$\min\limits_{J_j\in\mathrm{JB}_1}\{r_j\}+\Big\lceil\sum\limits_{i_1=1}^{z_1}P_{i_1}/m\Big\rceil$为合理的下界. 从工件集$\mathrm{JB}_1$中去掉到达时间最早的工件后得到工件集$\mathrm{JB}_2$,即$\mathrm{JB}_2=\{J_j|J_j\in\mathrm{JB}_1\text{ and }r_j>\min\limits_{J_j\in\mathrm{JB}_1}\{r_{j1}\}\}$. JB_2中工件的最小到达时间为$\min\limits_{J_j\in\mathrm{JB}_2}\{r_j\}$. 由于相对于$\mathrm{JB}_1,\mathrm{JB}_2$中的工件数减少了$\mathrm{JB}'_1=\{J_j|J_j\in\mathrm{JB}_1\text{ and }r_j=\min\limits_{J_j\in\mathrm{JB}_1}\{r_j\}\}$,基于$\mathrm{JB}_2$的真实下界必然不超过基于$\mathrm{JB}_1$的真实下界,即基于工件集$J$所得调度解的目标值不会小于基于$\mathrm{JB}_2$所得的下界. 由此表明,基于$\mathrm{JB}_2$的下界是原问题的一个下界. 通过文献[16]可计算出基于JB_2的下界值为$\min\limits_{J_j\in\mathrm{JB}_2}\{r_j\}+\Big\lceil\sum\limits_{i_1=1}^{z_2}P_{i_1}/m\Big\rceil$. 类似地,可以证明$\min\limits_{J_j\in\mathrm{JB}_w}\{r_j\}+\Big\lceil\sum\limits_{i_1=1}^{z_w}P_{i_1}/m\Big\rceil$为该问题的合理下界. 因此式(1)所示的LB为问题的一个有效下界. □

定理2　下界LB优于文献[16]所提下界LB_1.

证明　假定工件集合J中共有w种不同的到达时间,不妨设$r^1<r^2<\cdots<r^w$.

根据本文下界LB以及文献[16]中的下界LB_1的定义,LB可以表示为

$$\mathrm{LB}=\max\Big\{\max\{p_j+r_j\},\max_{v=1}^{w}\Big\{\min_{J_j\in\mathrm{JB}_v}\{r_j\}+\Big\lceil\Big(\sum_{i_1=1}^{z_v}P_{i_1}\Big)\Big/m\Big\rceil\Big\}\Big\}.$$

根据算法LB,有

$$\mathrm{JB}_v=\Big\{J_j|J_j\in\mathrm{JB}_{v-1}\text{ and }r_j>\min_{J_{j1}\in\mathrm{JB}_{v-1}}\{r_{j1}\}\Big\},$$

而$\min\limits_{J_j\in\mathrm{JB}_v}\{r_j\}=r^v$. 这里令

$$F(v)=\Big\lceil\Big(\sum_{i_1=1}^{z_v}P_{i_1}\Big)\Big/m\Big\rceil,$$

则下界LB可以简写为

$$\mathrm{LB}=\max\Big\{\max\{p_j+r_j\},\max_{v=1}^{w}\{r^v+F(v)\}\Big\}.$$

当$v=1$时,有

$$\mathrm{JB}_1=J,\ r^1+F(1)=\min_{J_j\in J}\{r_j\}+\left\lceil\sum_{i_1=1}^{z}P_{i_1}\Big/m\right\rceil,$$

此时

$$\mathrm{LB}_1=\max\{\max\{p_j+r_j\},r^1+F(1)\},$$

则

$$\mathrm{LB}=\max\left\{\max\{p_j+r_j\},r^1+F(1),\max_{v=2}^{w}\{r^v+F(v)\}\right\}=\max\left\{\mathrm{LB}_1,\max_{v=2}^{w}\{r^v+F(v)\}\right\}.$$

因此,可以看出本文所提下界LB不低于下界LB_1.

当存在任意一个工件J_a时,工件的到达时间为$r^v,1\leqslant v\leqslant w$,那么$r^v\geqslant r^1\geqslant 0,F(1)\geqslant F(v)\geqslant 1,\max\{r_j+p_j\}\geqslant 1$.

比较$r^v+F(v)$与$\max\{r_j+p_j\}$的大小有两种情况:

1) 当$\max\{r_j+p_j\}\geqslant r^v+F(v)$时,$\mathrm{LB}=\mathrm{LB}_1$.

2) 当$\max\{r_j+p_j\}<r^v+F(v)$时,进一步比较$r^v+F(v)$与$r^1+F(1)$的大小,则有下面两种情况:

i) 当$r^1+F(1)\geqslant r^v+F(v)$时,$\mathrm{LB}=\mathrm{LB}_1$;

ii) 当$r^1+F(1)<r^v+F(v)$时,则LB显然优于LB_1.

综上所述,下界LB不小于LB_1,且当存在工件使得$r^1+F(1)<r^v+F(v)$且$\max\{r_j+p_j\}<r^v+F(v)$时,LB优于LB_1. 说明下界LB优于下界LB_1. □

2 基于ACO的元启发式算法

蚁群算法是智能优化算法领域的研究热点之一,已被广泛应用于求解多种NP难的离散优化问题. 近年来,ACO算法因为具有构建性的特点也被用于求解批调度问题,其优势在于能够一次性构建出可行解,从而避免将不可行解转为可行解的处理. 在将ACO算法应用于求解调度问题时,蚁群利用代表群体间信息共享的信息素和个体的启发式信息来指导搜索,通过动态选择解的各个成分来构造高质量的解,同时确保解的可行性.

2.1 选择策略

蚁群算法在构建批时,通常采用无约束首工件选择策略,即从未分配工件集中随机选择一个工件,然后根据状态转移概率从候选列表中选取下一个工件放入批中[16-17]. 此外,也采用强约束首工件选择策略,即从未加工工件中选择某个特定工件作为批中的第一个工件.

当批中加入第一个工件后,为了降低批的浪费空间,随后加入批中的工件要尽量与批中已有工件的到达时间和加工时间相近. 由于后放入批中的工件受到之前放入批中的工件制约,第一个工件的选择对于批的特征有重要影响. 本文提出对放入批中首个工件的选择范围进行弱约束,并形成约束后的工件集合,即在选定的部分工件集合中选择一个工件作为建批时的第一个工件,随后生成批. 基于弱约束选择需要解决两个问题,即建立第一个工件集合FL和从FL中选择一个工件. 为了便于选择,为可选工件集中的每个工件J_j增加一个表示选择期望度的属性值d_j.

1) 可选集策略. 找到未分配工件的相对到达时间r'_j的最小值r',将相对到达时间在$[r',r'+\mathrm{WH_}T]$之间的工件放入集合中,其中WH_T表示工件选择范围的延伸参数,WH_T越大,工件选择范围越大,而WH_$T=0$表示只选择相对到达时间最短的工件. 其中当$r_j>C_i$时,$r'_j=r_j-C_i$;否则$r'_j=0$.

2) 首工件选择策略. 与传统的随机选择每个空批的首个工件不同,本文基于如下定义的工件选择概率fp从集合FL中选择加入空批的第一个工件

$$fp_j=\frac{d_j}{\sum\limits_{J_x\in\mathrm{FL}}d_x},\tag{2}$$

其中d_j表示工件被选择为首工件的期望值. 根据为空批选择首个工件时是否考虑工件的工件尺寸的不同情况,本文设计两种不同的首工件选择策略,并有两种对应的选择期望设计.

i) LP(Larger job size with higher probability)策略. 其中,$d_j=\max\{s_j-(r'_j-r'),0\}$,即$d_j$正比于该工件的工件尺寸,工件尺寸大的工件的选择期望值较高,这使得工件尺寸较大的工件被优先选择加工,以获得目标值更好的解.

ii) UP(Uniform probability)策略. 其中d_j与工件的工件尺寸无关,被定义为常量1.

在弱约束机制下,根据LP和UP策略,可以设计出两种首工件选择算法,即算法FSL(First job selection by LP)和FSU(First job selection by UP). 算法FSL描述如下.

算法FSL.

Step 1: 根据机器容量约束,从满足未调度工件尺寸约束的机器中选择完工时间最小的机器M_i,并获得机器的完工时间C_i.

Step 2: 根据式(2)找出工件尺寸不大于机器M_i容量的未调度工件中最小相对到达时间r',然后将相对到达时间在区间$[r',r'+\mathrm{WH_}T]$的满足机器容量

的工件放入集合FL.

Step 3: 根据LP选择策略,对FL中每个工件的期望值d_j进行赋值.

Step 4: 根据式(3),计算FL中工件的选择概率fp_j.

Step 5: 基于轮盘赌从FL中选择一个工件.

Step 6: 输出选择的机器M_i和工件J_j,结束.

算法FSU与算法FSL的区别仅在Step 3. 在FSU的Step 3中,按UP策略将FL中每个工件的选择期望d_j设为常量1.

2.2　启发式信息

启发式信息是构建解的重要信息,它通常与工件加入批中的浪费空间有关. 本文基于批的浪费空间及工件和批的匹配度分别设计启发式信息.

2.2.1　浪费空间启发式信息

定义1　解中m台平行机的浪费空间WS由批间浪费空间WS_{p_1}、批内浪费空间WS_{p_2}和机器完工时间不一致所导致的浪费空间WS_{p_3}三部分组成,其计算公式为

$$\mathrm{WS}=C_{\max}\sum_{i=1}^{m}S_i-\sum_{j=1}^{n}s_j\cdot p_j. \tag{3}$$

命题1　极小化一个批序列A最大完工时间等价于极小化A的浪费空间WS_A.

由命题1可知,为了使批序列A的$C_{\max}$最小,必须降低该序列的浪费空间WS_A,而尽量减少每台机器的浪费空间将有助于减少总的浪费空间.

假定机器M_i上有$b-1$个批$B_{1i},B_{2i},\cdots,B_{(b-1)i}$,若在批$B_{(b-1)i}$之后插入批$B_{bi}$,则$M_i$上增加的浪费空间为$S_i(C_{bi}-C_{(b-1)i})-\sum\limits_{J_j\in B_{bi}}s_j\cdot p_j$. 此时机器$M_i$的浪费空间(如图1所示)为

$$\mathrm{WS}_{bi}=\mathrm{WS}_{(b-1)i}+S_i(C_{bi}-C_{(b-1)i})-\sum_{J_j\in B_{bi}}s_j\cdot p_j. \tag{4}$$

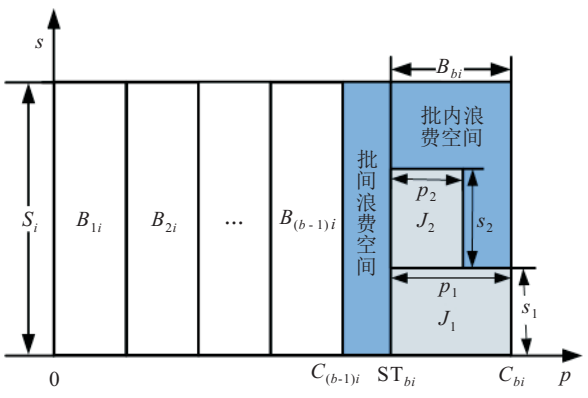


图1　机器M_i的浪费空间

当某个工件J_j放入批B_{bi}后,M_i的浪费空间为

$$\mathrm{WS}'_{bi}=\mathrm{WS}_{(b-1)i}+S_i(C'_{bi}-C_{(b-1)i})-\sum_{J_j\in B_{bi}}s_j\cdot p_j-s_j\cdot p_j. \tag{5}$$

其中:$C_{bi}=\mathrm{ST}_{bi}+P_{bi}$,$C'_{bi}=\mathrm{ST}'_{bi}+P'_{bi}$. 浪费空间的变化为

$$\Delta\mathrm{WS}_{bi}=\mathrm{WS}_{bi}-\mathrm{WS}'_{bi}=S_i(C_{bi}-C'_{bi})+s_j\cdot p_j. \tag{6}$$

因而基于M_i浪费空间的变化,定义如下启发式信息:

$$\eta^1_{ibj}=\begin{cases}\Delta\mathrm{WS}_{bi}+1, & \Delta\mathrm{WS}_{bi}>0;\\ 1, & \Delta\mathrm{WS}_{bi}\leqslant 0.\end{cases} \tag{7}$$

2.2.2　工件匹配度启发式信息

考虑到工件到达时间和工件尺寸对解的影响,尽量将到达时间r_j和加工时间p_j与批开工时间ST_{bi}和批加工时间P_{bi}相近的工件放入同一个批中,提出基于工件匹配度的启发式信息如下:

$$\eta^2_{ibj}=1/(\mathrm{dr}+1)+1/(\mathrm{dp}+1). \tag{8}$$

其中:dr、dp分别为工件到达时间匹配度和工件加工时间匹配度. 若$r_j>\mathrm{ST}_{bi}$,则$\mathrm{dr}=r_j-\mathrm{ST}_{bi}$,否则$\mathrm{dr}=0$;若$p_j>P_{bi}$,则$\mathrm{dp}=p_j-P_{bi}$,否则$\mathrm{dp}=P_{bi}-p_j$.

2.3　候选列表

为了缩小搜索范围,以当前部分解的邻近解为候选列表,提高算法的搜索性能. 考虑到每个工件只能放入一个批中,候选表中的工件必须是未分配的工件. 其次,考虑到批中剩余空间限制,候选表中的工件尺寸不能大于批中剩余空间的尺寸. 再次,根据式(6),当候选表中的工件加入批后,$\Delta\mathrm{WS}_{bi}>0$可以降低批的浪费空间

综合以上3点,最终的候选列表CL定义如下:

$$\mathrm{CL}_{bi}=\Big\{J_j|s_j\cdot p_j>S_i(\mathrm{ST}'_{bi}-\mathrm{ST}_{bi})+S_i(P'_{bi}-P_{bi})\ \text{and}\ J_j\in J\ \text{and}\ s_j\leqslant S_i-\sum_{J_x\in B_{bi}}s_x\Big\}. \tag{9}$$

其中:ST'_{bi}为批B_{bi}加入工件J_j后批的加工时间,若$r_j>\mathrm{ST}_{bi}$,则$\mathrm{ST}'_{bi}=r_j$,否则$\mathrm{ST}'_{bi}=\mathrm{ST}_{bi}$;$P_{bi}$和$P'_{bi}$分别为放入工件$J_j$前后批的加工时间,若$p_j>P_{bi}$,则$P'_{bi}=p_j$,否则$P'_{bi}=P_{bi}$.

2.4　信 息 素

2.4.1　信息素定义

本文采用一个$n\times n$的矩阵Φ来存储信息素,其任一元素φ_{xy}表示工件J_x和J_y分配在同一批中的期

望值,初始化为$((1-\rho)\mathrm{LB})^{-1}$,$\rho$是介于0与1之间的参数,LB是由算法LB所得下界. 信息素τ_{ibj}为将工件J_j分配在批B_{bi}中的期望度,定义如下:

$$\tau_{ibj}=\sum_{J_x\in B_{bi}}\varphi_{xj}/|B_{bi}|. \tag{10}$$

2.4.2 信息素更新

蚁群算法在每代结束后,需对信息素进行更新. 本文算法中采用当代最优解来更新信息素. $\varphi_{xy}(t)$表示第t代J_x和J_y分配在同一批中的期望值.

$$\varphi_{xy}(t+1)=(1-\rho)\varphi_{xy}(t)+m_{xy}(t)\Delta\varphi_{xy}(t). \tag{11}$$

其中:ρ为信息素挥发率,$m_{xy}(t)$表示第t代J_x和J_y分配在同一批中的次数. 若$J_y\in B$,则$\Delta\varphi_{xy}(t)=Q/C'_{\max}$,否则$\Delta\varphi_{xy}(t)=0$. 其中:$C'_{\max}$为当代最优解目标值,$Q$为输入参数.

2.5 解的构建

为了直观地描述解的构建过程,这里引入一个初值为$[1,1,\cdots,1]$的n维的调度信息表TB,表示初始时无工件被调度;一旦某个工件被调度,则将TB中该工件相应信息位更新为0;$\mathrm{TB}=[0,0,\cdots,0]$则表示所有工件均被调度.

在机器容量限制下,工件J_j的尺寸不能超过加工该工件的机器容量,而本文所研究问题中部分工件的尺寸超过部分机器容量,因而根据机器容量将工件分成h个工件子集,记为$J^1,J^2,\cdots,J^h$,即

$$J=J^1\bigcup J^2\bigcup\cdots\bigcup J^h.$$

其中

$$J^1=\{J_j|s_j\leqslant S^1\},\ J^2=\{J_j|S^1<s_j\leqslant S^2\},$$
$$J^3=\{J_j|S^2<s_j\leqslant S^3\},\cdots,$$
$$J^h=\{J_j|S^{h-1}<s_j\leqslant S^h\}.$$

大尺寸工件只能放入大容量机器中加工,而小尺寸工件不但可以放入小容量机器,也可以放入大容量机器. 这就造成了大容量机器的稀缺性. 所以从候选列表中选择工件时,如果所在机器为大容量机器,则优先考虑大容量等级的工件.

根据2.1节的首个工件选择策略FSL和FSU,相应地有两种构建解的算法,记为算法CLL(Candidate list-FSL based algorithm)和算法CLU(Candidate list-FSU based algorithm). 在算法CLL中调用FSL来确定空批所在的机器和放入空批中的第一个工件,其过程描述如下.

算法CLL.

Step 1: 初始化工件状态信息表TB$=[1,1,\cdots,1]$.

Step 2: 如果$\mathrm{TB}\neq[0,0,\cdots,0]$,则调用算法FSL选择构建批的机器$M_i$和第一个放入批的工件;否则,输出结果,结束.

Step 3: 机器M_i为第k种容量的机器,即$M_i\in M^k$,M^k中所有机器容量均为S^k. 在机器上构建一个空批,该空批的开始加工时间为该机器的完工时间C_i,将Step 2选择的工件放入空批,更新调度信息表TB.

Step 4: 根据候选列表式(9)构建当前批的候选列表CL_{bi}.

Step 5: 当$\mathrm{CL}_{bi}=\varnothing$时,批构建完成,更新机器$M_i$的完工时间$C_i$,转Step 2;否则,转到Step 5.1. C_i的更新公式如下:

$$C_i=\max\{C'_i,R_{bi}\}+P_{bi}. \tag{12}$$

其中:R_{bi}为批的到达时间,P_{bi}为批的加工时间,C'_i为未加入批的机器完工时间.

Step 5.1: 令$\mathrm{JL}_{ibk}=\{J_j\in\mathrm{CL}_{bi}\text{ and }J_j\in J^k\}$,如果$\mathrm{JL}_{ibk}\neq\varnothing$,则根据概率$p_{iby}$从集合$\mathrm{JL}_{ibk}$中选择工件$J_y$加入批中,其中

$$p_{iby}=\begin{cases}\dfrac{\tau_{iby}(\eta^1_{iby})^{\alpha}(\eta^2_{iby})^{\beta}}{\displaystyle\sum_{J_x\in\mathrm{JL}_{ibk}}\tau_{ibx}(\eta^1_{ibx})^{\alpha}(\eta^2_{ibx})^{\beta}},\ J_y\in\mathrm{JL}_{ibk};\\ 0,\ \text{otherwise.}\end{cases} \tag{13}$$

Step 5.2: 如果$\mathrm{JL}_{ibk}=\varnothing$,则$k--$,转Step 5.1.

Step 5.3: 更新调度信息表TB和批当前尺寸,转Step 4.

算法CLU与CLL的区别仅在于Step 2. 在CLU的Step 2中,调用FSU来选择构建空批的机器M_i和第一个放入该批的工件.

2.6 局部优化

每只蚂蚁构建完解后,采用局部优化策略LO对解进行改进,以进一步提高解的质量.

算法LO.

Step 1: 提取工件集合$\mathrm{EJ}=\varnothing$;找出M中完工时间最小的机器,记为$C_{\min}$;对于M中每个机器,从最后一个批开始提取批直到该机器的完工时间小于等于$C_{\min}$;将所提取批中的工件放入集合EJ中.

Step 2: 将M中的机器按照当前完工时间非减排序,为每台机器构建满足机器容量的候选列表$\mathrm{ML}_i=\{J_j|J_j\in\mathrm{EJ}\text{ and }s_j\leqslant S_i\}$.

Step 3: 选择机器序列中第一个候选列表$\mathrm{ML}_i\neq\varnothing$的机器$M_i$,并在$M_i$上建立一个空批$B_{bi}$.

Step 3.1: 将 ML_i 中的工件按照规则 LPT(Longest processing time) 排序.

Step 3.2: 从 ML_i 中按照 FF(First fit) 规则选择工件加入批 B_{bi} 中, 直到批满, 并根据式 (12) 更新机器完工时间 C_i.

Step 4: 将 EJ 中已调度的工件移出工件集合 EJ. 若 EJ 中有工件未被调度, 则跳转 Step 2; 否则, 输出解, 结束.

2.7　算法描述

基于前面的讨论, 在弱约束机制下构建解时, 针对为空批选择首个工件时是否考虑工件的工件尺寸, 设计两种不同的解构建算法, 即 CLL 和 CLU, 对应的蚁群优化算法分别记为 ACOL(CLL based ACO algorithm) 和 ACOU(CLU based ACO algorithm). 其中: 算法 ACOL 中调用算法 CLL 来构建解, 而算法 ACOU 则调用 CLU 构建解. 算法 ACOL 描述如下.

算法 ACOL.

Step 1: 初始化参数: 最大迭代次数 $T_{\max}$, 工件数 Q, 信息素蒸发率, 启发式权值, 蚂蚁数.

Step 2: 根据式 (1), 计算下界 LB.

Step 3: 当前代数 $t=0$, 计算初始信息素.

Step 4: 若 $t \geqslant T_{\max}$, 则输出全局最优解并结束.

Step 5: 调用算法 CLL 为每只蚂蚁构建解, 并根据式 (11) 更新信息素.

Step 6: 调用算法 LO 优化每只蚂蚁构建的解.

Step 7: 更新局部最优解和全局最优解.

Step 8: $t=t+1$, 转 Step 4.

算法 ACOU 与 ACOL 的区别仅在于 Step 5 中调用 CLU 构建解, 并根据式 (11) 更新信息素.

3　仿真实验

3.1　实验设计

为了证明本文提出算法的有效性, 采用文献 [18] 提出的随机生成测试实例对算法进行测试. 根据不同的工件生成 6 组实验实例, 工件数分别是 90, 108, 126, 144, 162, 180, 每组包含 10 个测试实例. 工件加工时间 p_j 均匀分布于 [8, 48]. 工件到达时间 r_j 均匀分布于 $[0, L]$. 机器总数设为 10, 机器容量有 3 种, 分别为 10, 25, 65. 在实际应用中, 容量大的机器通常价格较高, 对应机器数相对较少, 所以实验中将 3 种容量的机器数分别设置为 5, 3, 2. 根据机器容量约束, 工件可分为 3 类, 即 $J=J^1 \bigcup J^2 \bigcup J^3$, 其中 J^1 中的工件可以放入所有机器中, J^2 中的工件可以放入机器集合 M^2、M^3 中, J^3 中的工件只能放入机器集合 M^3 中. $J^1 \sim J^3$ 的工件数分别设置为 $2n/3, 2n/9, n/9$, 以使 $|J^1|>|J^2|>|J^3|$.

本文基于泊松分布生成工件尺寸[16], 即 $\{s_j|J_j \in J^1\} \sim P(\lambda_1)$, $\{s_j|J_j \in J^2\} \sim P(\lambda_2)+10$, $\{s_j|J_j \in J^3\} \sim P(\lambda_3)+25$. 其中: $\lambda_1=5, \lambda_2=12.5, \lambda_3=32.5$. 这使得同一组中的小尺寸工件的数量较大尺寸工件多, 以使更多小尺寸工件分配到较大尺寸工件所在的批中, 从而使调度过程更加复杂. 考虑到工件尺寸边界需要与机器容量相容, 例如工件 $J_j \in J^k$. 如果该工件尺寸 s_j 超过 M^k 中机器的上界 S^k, 则 $s_j=S^k$; 如果工件尺寸 s_j 小于机器下界 S^{k-1}, 则 $s_j=S^{k-1}$. 其中 $S^0=1$. 为了确保有足够的小尺寸工件来填充机器, 再分别从 $(S^{k-1}, S^k/2]$ 和 $(S^k/2, S^k]$ 中分别选取 70 % 和 30 % 的数据组成最终测试工件集, 其中当 $k=1$ 时, 分别从 $(0, S^k/2]$ 和 $(S^k/2, S^k]$ 中选取 70 % 和 30 % 的数据组成最终测试工件集.

工件的到达时间均匀分布在 [0,L] 之间, 其中 L 是通过算法 L 得出的.

算法 L.

Step 1: 对于 $\forall J_j \in J$, 将尺寸为 s_j 的工件单位化为 s_j 个单位尺寸、加工时间为 p_j、到达时间为 0 的工件, 这样可以得到一个新的单位尺寸工件集合 J'.

Step 2: 对于 $\forall M_i \in M$, 将容量为 S_i 的机器单位化为 S_i 台单位容量为 1 的单位机器, 这样得到一个新的单位容量的机器集合 M'.

Step 3: 将工件集合 J' 中的工件按 LPT 规则排序, 然后将集合 J' 中的工件依次安排到机器集合 M' 中当前完工时间最小的机器上加工, 最终得到的机器最大完工时间记为 L.

3.2　参数设置

蚁群算法的参数不仅影响算法的收敛速度, 而且影响搜索到的解质量, 因此为了获得较好的解, 有必要通过实验来确定适合的参数值. 影响 ACO 算法性能的参数包括蚂蚁数、迭代次数、启发式信息影响因子、信息素蒸发率、批中第一个工件选择集合范围参数 WH_T.

根据文献 [16] 和 [18] 确定的部分实验参数如下: $\mathrm{AntNum}=20, T_{\max}=200, \rho=0.5$ 和 $Q=n$. 此外, 影响算法性能的参数还有影响因子 α 和 β, 以及批中第一个工件选择范围参数 WH_T.

为了确定 α 和 β 的值, 首先将 WH_T 的值设置为 0, 然后从 6 组工件数不同的工件集中分别随机选择 3 个实例, 保持其他参数不变, 在这 18 个实例上采用不同组合的 α 和 β 值及算法 ACOL 进行测试. 参照文献 [16] 的方法, 将 α 和 β 的值分别设置在

[1,10]和[0,1]之间,针对9组不同的参数组合值$(1,1)$, $\left(1,\frac{1}{3}\right)$,(3,1), $\left(1,\frac{1}{5}\right)$,(5,1), $\left(1,\frac{1}{7}\right)$,(7,1), $\left(1,\frac{1}{9}\right)$,$(9,1)$进行测试,所得的平均目标值分别为198.45, 198.2, 197.26, 198.29, 197.1, 198.38, 197.11, 198.17, 196.78. 可以看出,当$\alpha=9$和$\beta=1$时,解的质量相对最好,因此后面的实验中均采用这组参数值.

为了确定WH_T的值,先设置$\alpha=9$以及$\beta=1$,并将WH_T的取值范围设为[0,14]. 为了进一步确定具体值,再分别从6组工件数不同的工件集中随机选择3个实例,分别将WH_T取值为$0,2,4,6,8,10,12,14$,使用算法ACOL在这18个实例上进行测试,保持其他参数值不变,所得结果分别为$0,-2.7,-4.6,-4.9,-4.6,-3.4,0.7,3.4$. 可以看出,当WH_$T$取4, 6, 8时,平均目标值的变化幅度较少;当超过10时,解的质量大幅度下降. 当WH_$T=6$时,解的质量最好,故后面实验中均采用此值.

3.3 实验结果与分析

为了验证本文弱约束下基于不同首工件选择策略的算法ACOL和ACOU的性能,将本文所提算法与文献[16]随机选择每个批首工件的ACO1算法(The first proposed ACO-based algorithm)、文献[12]的GA算法和文献[14]的PSO算法进行对比. 此外,为了验证局部优化算法LO的有效性,将算法ACOL的局部优化策略去掉,得到不含局部优化的UACOL(Unlocal optimization-CLL based ACO algorithm)算法以进行对比实验.

所有的算法均在Pentium(R) Dual-Core 2.8 GHz CPU,2 GB RAM环境下,采用C++编程实现. 为了客观比较不同算法的性能,所有的ACO算法的迭代次数和每代蚂蚁的总数均分别设为200和20,而PSO算法和GA的迭代次数均设置为500代,PSO算法的粒子数设置为60,GA的染色体数设置为100. 为了降低初始解对蚁群优化算法求解结果的影响,采用每个算法在每个实例上运行10次后所得解的平均目标值代表该算法在该实例上的结果. 一个算法在某个工件规模下的平均结果是该算法在该组工件规模下10个测试实例上的平均结果. 各算法的性能是通过求解所得的平均目标值与其对应下界的相对百分比来衡量的. 定义$R_{\rm A}$为算法A所得目标值与下界的距离,其计算公式为

$$R_{\rm A}=\left(\frac{C_{\max}^{\rm A}}{\rm LB}-1\right)\times 100\%. \tag{14}$$

其中:$C_{\max}^{\rm A}$为算法A所得解的制造跨度的平均值,$C_{\max}^{\rm A}$越小则$R_{\rm A}$越小,算法A所得解的目标值越接近下界,表明解的质量越高. 此外,定义ACOL算法对算法A的改进度为$R_{\rm A}-R_{\rm ACOL}$,以表示算法ACOL对PSO和GA的改进效果,其值越大,表明改进度越大.

针对不同工件规模问题的平均测试结果如图2 ~图5所示,分别表示各ACO算法的平均性能R、平均标准差SD和平均时间性能的对比结果,横坐标表示每组实例的工件规模,纵坐标表示各算法在每种工件规模下10个测试实例上的各性能指标的平均值.

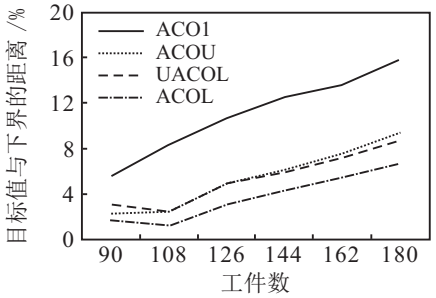


图2 各算法R值比较

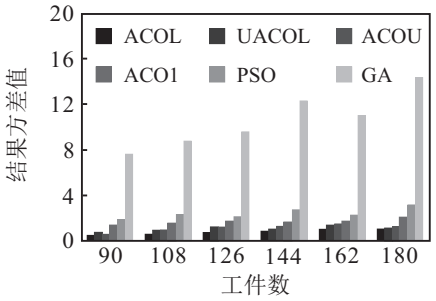


图3 SD值比较

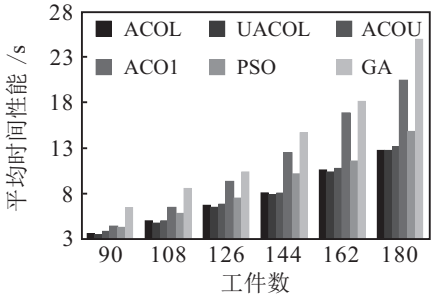
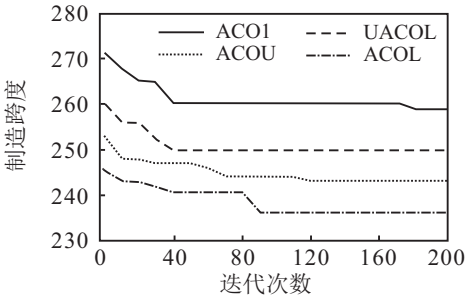
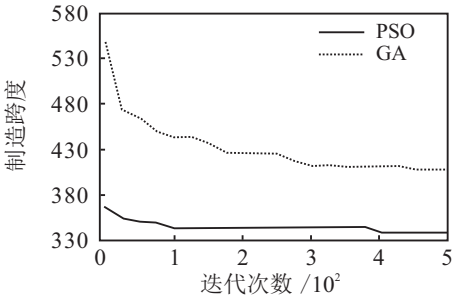


图4 各算法运行时间比较

从图2可以看出,虽然随着工件数量的增加,所有的ACO算法的R值都出现了逐渐增加的趋势,但ACOL算法在各测试集上所得解的平均质量均是最好的. 当工件数达到180时,ACOL所得解与下界的距离不到7 %. 在工件数较少如90和108时,ACOL的R值较小且保持平稳.采用弱约束机制的算法ACOL、UACOL和ACOU的解质量均优于无约束策略算法ACO1. 随着工件数的增加,弱约束策略的算法相对于无约束策略算法的优势越来越明显. 当工件规模为90时,ACOL的平均性能比ACO1的平均性能优越近4 %;而当工件规模达到180时,两者的



(a) 几种ACO算法



(b) GA和PSO算法

图5　算法收敛性能比较

差值接近9%. 此外, ACOL算法还优于算法UACOL, 说明使用局部优化策略可以有效提高解的质量. 算法ACOL与ACOU的区别在于选择首个工件的方法, 从图2所示ACOL和ACOU的平均性能可以看出, ACOL所得解的质量优于ACOU, 表明在首个工件的选择策略上, 使用工件尺寸大高概率选择策略较均等概率选择策略更加有效.

不同问题规模下, ACOL算法对PSO和GA算法的改进度分别为23.68, 29.32, 31.67, 34.25, 35.4,37.75和47.92, 59.85, 64.41, 73.35, 75.19, 80.53. 可以看出, 算法ACOL对PSO和GA均有显著优势, 对GA的改进效果相对于PSO更明显, 且随着工件规模的增大, ACOL算法的改进效果越明显. 当工件规模达到180时, ACOL算法对PSO、GA的改进度最大.

图3给出了各算法在不同问题规模下的标准差SD值, 其中SD值越低表示解的质量越稳定, 算法的鲁棒性越好. 可以看出, 随着工件规模的增大, 各算法的SD值均有增大趋势, 说明算法的鲁棒性随着问题规模的增大出现下降趋势. 这是因为问题规模的增加导致解空间的增大, 也增加了问题的求解难度, 从客观上会降低算法的鲁棒性, 而相比于无约束策略的算法, 使用弱约束策略的算法鲁棒性更好, 说明弱约束策略可以有效缩小搜索空间, 使得蚁群可以深度挖掘有可能含有最优解的解空间, 从而减少无效搜索, 提高算法整体的搜索性能. 此外, 可以看出, ACOL的鲁棒性也优于ACOU, 表明工件尺寸大高概率选择策略有效提高了蚁群搜索到优质解的概率. ACO算法的鲁棒性均优于PSO和GA, 这是由于ACO算法在构建解的过程中通过建立候选列表缩小了搜索空间, 通过工件概率选择方式增加了蚁群搜索到优质解的概率.

图4比较了各算法的时间性能. 可以看出, ACOL相比未使用局部优化策略的算法UACOL, ACOL耗时更长. 这表明局部优化策略会显著增加算法计算时间. 此外, 还可以看出, 由于局部优化策略所降低的时间性能均在0.5 s内, 所增加的搜索时间是可以接受的. ACOL由于使用弱约束策略, 在建批时选择首个工件时需要额外消耗时间. ACO1虽然不需要对批中首个工件进行选择约束, 但是该算法形成批后, 需要对批进行排序, 并将批按照顺序安排到机器中, 且由于ACO1形成的批序列更加复杂, 需要耗费更多的时间实现对结果的局部优化. 此外, ACO1在对工件按尺寸分类的过程中也需要消耗一定时间. 另外, 还可以明显看出, 随着问题规模的增加, 使用弱约束策略比无约束策略消耗的时间更少, 且运行时间上的优势也随着问题规模的增加而更明显. PSO和GA在运行中均消耗较多的计算时间, 其中GA在所有算法中消耗的时间最多, 除了算法本身因素外, 其原因还在于本次实验为PSO和GA算法设置了较多的迭代次数和较大的种群规模以确保获得质量较优的解.

图5给出了各算法求解一个随机选择的工件规模为180的实例的收敛趋势比较, 其中横坐标表示算法的迭代代数, 纵坐标表示问题的目标值$C_{\max}$. 因为4个ACO算法所得解的目标值远小于PSO和GA算法所得解, 因此将原图分为(a)和(b)两个子图, 其中图5(a)给出了4个ACO算法的收敛趋势, 图5(b)给出了GA和PSO算法的收敛趋势. 从图5(a)可以看出, 算法ACOL的初始种群就获得了质量较好的解, 这是因为相比其他算法, ACOL的弱约束策略和工件尺寸大的工件高概率选择策略, 使得蚁群始终可以搜索到较高质量的解, 同时也提高了算法的收敛性. 相比于GA和PSO算法, ACO算法由于具备构造型特征, 在构造解的过程中通过候选列表、概率选择等方式, 提高了收敛性. 从图5(b)可以看出, GA的收敛性能比PSO更差, 相比于ACO算法和PSO算法, 500次迭代不能使得GA获得更高质量的解.

综上所述, 可以看出, 本文所提ACOL算法的综合性能最优, 一方面是因为弱约束策略可以有效缩小解的搜索范围, 而工件尺寸大高概率选择策略将高概率搜索到更感兴趣的解空间, 从而提高算法的搜索性能及求解质量; 另一方面是因为启发式信息、候选列表的构建, 均有利于提高算法的搜索性能, 而局部优

化策略则进一步提高了解的质量.

4 结 论

本文主要研究在容量不同的平行批处理机上,尺寸差异的工件动态到达的批调度问题. 首先,分析问题特征,给出问题下界并证明其有效性;然后,提出在批形成过程中首个放入批中工件的弱约束策略. 根据弱约束策略,确定了首个工件候选集合,并提出两种在候选集合中选择首工件的工件选择策略;最后,根据不同的工件选择策略,提出了两种弱约束下的蚁群调度算法来求解该问题. 仿真实验结果表明,本文中所提的批中首个工件弱约束策略优于无约束策略,而且在选择工件时,工件尺寸大的工件高概率选择策略优于平均概率选择策略.

下一步的研究可以从以下几个方面进行:一是将本文的算法推广到更复杂的调度模型中,比如机器加工速度不同、工件不相容等约束条件;二是在首个工件选择策略上进一步改进,提出性能更好的工件选择策略.

参考文献(References)

[1] Ikura Y, Gimple M. Efficient scheduling algorithms for a single batch processing machine[J]. Operations Research Letters, 1986, 5(2): 61-65.

[2] Uzsoy R. Scheduling a single batch processing machine with non-identical job sizes[J]. Int J of Production Research, 1994, 32(7): 1615-1635.

[3] Dupont L, Ghazvini F. Minimizing makespan on a single batch processing machine with non-identical job sizes[J]. European J of Automation, 1998, 32(4): 431-440.

[4] Dupont L, Dhaenens-Flipo F. Minimizing the makespan on a batch machine with non-identical job sizes: An exact procedure[J]. Computers & Operations Research, 2002, 29(7): 807-819.

[5] Melouk S, Damodaran P, Chang P. Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing[J]. Int J of Production Economics, 2004, 87(2): 141-147.

[6] Kashan H, Karimi B, Jolai F. Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes[J]. Int J of Production Research, 2006, 44(12): 2337-2360.

[7] Chou F D, Chang P C, Wang H M. A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem[J]. Int J of Advanced Manufacturing Technology, 2006, 31(3/4): 350-359.

[8] 王栓狮, 陈华平, 程八一, 等. 一种差异工件单机批调度问题的蚁群优化算法[J]. 管理科学学报, 2009, 12(6): 78-82.
(Wang S S, Chen H P, Cheng B Y, et al. Minimizing makespan on a single batch processing machine with nonidentical job sizes using ant colony optimization[J]. J of Management Sciences, 2009, 12(6): 78-82.)

[9] 许瑞, 陈华平, 邵浩, 等. 极小化总完工时间批调度问题的两种蚁群算法[J]. 计算机集成制造系统, 2010, 16(6): 1255-1264.
(Xu R, Chen H P, Shao H, et al. Two kinds of ant colony algorithms to minimize the total completion time for batch scheduling problem[J]. Computer Integrated Manufacturing Systems, 2010, 16(6): 1255-1264.)

[10] Xu S, Bean J. A genetic algorithm for scheduling parallel non-identical batch processing machines[C]. Proc of IEEE Symposium on Computational Intelligence in Scheduling. Honolulu: IEEE Press, 2007: 143-150.

[11] Cheng B, Wang Q, Yang S, et al. An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes[J]. Applied Soft Computing, 2013, 13(2): 765-772.

[12] Wang H, Chou F. Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics[J]. Expert Systems with Applications, 2010, 37(2): 1510-1521.

[13] Chen H, Du B, George H. Metaheuristics to minimize makespan on parallel batch processing machines with dynamic job arrivals[J]. Int J of Computer Integrated Manufacturing, 2010, 23(10): 942-956.

[14] Damodaran P, Diyadawagamage D, Ghrayeb O, et al. A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines[J]. Int J of Advanced Manufacturing Technology, 2012, 58(9): 1131-1140.

[15] Zhou S, Liu M, Chen H, et al. An effective discrete differential evolution algorithm for scheduling uniform parallel batch processing machines with non-identical capacities and arbitrary job sizes[J]. Int J of Production Economics, 2016, 179: 1-11.

[16] Jia Z, Li X, Leung J. Minimizing makespan for arbitrary size jobs with release times on P-batch machines with arbitrary capacities[J]. Future Generation Computer Systems, 2017, 67: 22-34.

[17] 许瑞, 陈华平. 含不同到达时间和尺寸的批调度优化算法[J]. 计算机集成制造系统, 2011, 17(9): 1944-1953.
(Xu R, Chen H P. Optimization algorithm on batch scheduling with different release time and non-identical job sizes[J]. Computer Integrated Manufacturing Systems, 2011, 17(9): 1944-1953.)

[18] Wang J, Leung J. Scheduling jobs with equal-processing-time on parallel machines with non-identical capacities to minimize makespan[J]. Int J of Production Economics, 2014, 156: 325-331.

（责任编辑：闫 妍）