

基于目标增量的无等待流水调度快速迭代贪婪算法

朱 夏 李小平 王 茜

(东南大学计算机科学与工程学院 南京 210096)

(计算机网络和信息集成教育部重点实验室 南京 210096)

摘 要 最小化总完工时间无等待流水调度是典型的 NP-完全问题,广泛存在于实际生产系统. 改变传统求解调度序列目标函数的模式,提出目标增量法,通过目标函数变化量判断新解的优劣,大大降低算法所需计算时间;通过证明启发式算法基本操作的目标增量性质,设计两种基本目标增量法以快速评估新产生解的质量. 提出快速迭代贪婪算法 FIG(Fast Iterative Greedy algorithm)求解该问题,构造初始解生成算法,提出分段式重构局部搜索方法和迭代改进全局搜索策略以进一步提高解的质量. 基于 110 个经典 Benchmark 实例,将提出的 FIG 算法与目前求解该问题较好的启发式算法 PH1p 和元启发式算法 SRTS、DPSOvnd 进行比较,实验结果表明 FIG 在性能上优于 SRTS 和 PH1p,略逊于 DPSOvnd;在效率上优于 SRTS 和 DPSOvnd,略逊于 PH1p.

关键词 无等待流水调度;目标增量;启发式算法;总完工时间最小

中图法分类号 TP278 **DOI 号**: 10.3724/SP.J.1016.2009.00132

Objective Increment Based Iterative Greedy Heuristic for No-Wait Flowshops with Total Flowtime Minimization

ZHU Xia LI Xiao-Ping WANG Qian

(School of Computer Science and Engineering, Southeast University, Nanjing 210096)

(Key Laboratory of Computer Network and Information Integration of Ministry of Education, Southeast University, Nanjing 210096)

Abstract No-wait flowshops with total flowtime minimization are typical NP-Complete combinatorial optimization problems, widely existing in practical manufacturing systems. Different from traditional methods in which objectives are completely computed for a new generated schedule, objective increment methods are presented in this paper. Whether a new schedule is better or worse than the original one is judged just by the objective increment, which can reduce computational time considerably. Objective increment properties are analyzed for fundamental operations of heuristics. Based on the properties, two fundamental methods are introduced for fast evaluating schedules. FIG (Fast Iterative Greedy algorithm) is proposed for the considered problem, which includes initial solution generating and solution improvement phases. Besides an initial solution generating method being constructed, a segment based reconstructive heuristic and an iterative improvement procedure are developed for local and global search respectively to improve the current solution. FIG is compared with PH1p, SRTS and DPSOvnd algorithms on 110 traditional benchmark instances. Computational results show that FIG outperforms SRTS and PH1p but a little worse than DPSOvnd in effectiveness. In efficiency, FIG is better than SRTS and DPSOvnd but a little worse than PH1p.

Keywords no-wait flowshops; objective increment; heuristic; total flowtime minimization

收稿日期:2007-10-16;最终修改稿收到日期:2008-12-07. 本课题得到国家自然科学基金(60504029,60672092)和国家“八六三”高新技术研究发展计划项目基金(2008AA04Z103)资助. 朱 夏,女,1982 年生,博士研究生,主要研究方向为调度优化、服务计算. E-mail: zhuxia@seu.edu.cn. 李小平,男,1970 年生,博士,副教授,博士生导师,主要研究领域为调度优化、服务计算、企业互操作. 王 茜,女,1946 年生,教授,博士生导师,主要研究领域为信息集成技术、数据库技术以及计算机协同工作.

1 引言

调度(排序)是制造和服务等行业中一类重要且普遍存在的问题. 调度是为一系列任务在不同机器上安排一个合理的加工时间表, 达到某个或某些性能最优, 如最小化最大完工时间(makespan)、总完工时间(total flowtime)、延迟(tardiness)等^[1]. 无等待流水调度问题是一类重要的约束流水调度问题, 它广泛存在于流程式企业^[2], 如化工制造、钢铁铸造、食品加工、塑料塑造等. 此外, 高级制造环境如 Just-in-time、柔性制造系统以及机器人之间具有高度协作的加工环境通常也被模型化为无等待调度.

所有任务的完工时间之和称为总完工时间(total flowtime), 它是调度的一个重要评价指标. 总完工时间最小可促使资源稳定有效地利用、任务的快速周转和在制品库存最小^[3]. 最小化总完工时间的无等待流水调度可表示为 $F_m | nwt | \sum C_i$ ^[4]. Hall 和 Sriskandarajah^[2]、MacCarthy 和 Liu^[5] 对该问题的现状做了较为详细的综述, 并指出该问题即使在两台机器的情况也是 NP-完全的. Van Deman 和 Baker^[6] 提出分枝定界法来寻求最优方案, 并提出了一系列过程来产生最优值的下界. Adiri 和 Pohoryles^[7] 证明了 2-机问题具有最优调度的一些性质, 并给出一些定理作为多项式有界求解具有升序或降序占优机器序列的 m -机问题的基础. Van der Veen 和 Van Dal^[8] 指出当目标函数限于半序的加工时间矩阵时该问题是可解的. Rajendran 和 Chaudhuri^[9] 提出了构造启发式算法 RC1 和 RC2, 实验结果表明对于任务数在区间 $[12, 20]$ 的小规模问题, RC1 优于 RC2, 且二者性能明显优于 Bonney 和 Gundry^[10]、King 和 Spachis^[11] 的算法. Bertolissi^[12] 提出了基于成对比较和任务插入的构造启发式算法 BE, 实验结果表明 BE 优于 RC1 和 RC2^[9]、Bonney 和 Gundry^[10] 的算法. Chen 等^[13] 提出了采用启发式算法与随机生成方式相结合产生初始种群的遗传算法(简称 TGA), 实验结果表明 TGA 在 200 个实例中有 168 个结果比 RC1 好, 但在问题规模较大时并不稳定. Fink 和 Voß^[14] 以 Taillard^[15] 的 110 个经典 Benchmark 实例为基础, 分析比较了多种元启发式算法, 表明以 Chins 或 Pilot-10 产生初始解、移位与自适应禁忌搜索相结合的方法(简称 SRTS)是有效的算法. Aldowaisan 和 Allahverdi^[16] 构造了 6 个

复合启发式算法 PH1(p) ~ PH4(p), 并将其与 RC1, RC2^[9], BE^[12] 和 TGA^[13] 进行比较, 结果表明 PH1p 的性能明显优于其它算法. Kumar 等^[17] 提出了比 PH1p 性能更优的心理学克隆选择算法, 但该算法难于实现. Pan Quan-Ke^[18] 提出基于 VND 局部搜索和离散微粒群的算法 DPSOvnd, 将速度与位置更新策略投影为离散空间的进化算子, 并采用任务交换和插入操作加速计算方法, 该算法是目前性能最好的算法.

本文基于文献[19]求解 $F_m | \sum C_i$ 问题的迭代贪婪算法 IG, 构建了一个求解 $F_m | nwt | \sum C_i$ 问题的目标增量的快速迭代贪婪算法 FIG. 根据问题的特点构造初始解; 通过将序列随机划分为若干子序列并分别采用分解与重构操作产生邻域, 扩大局部搜索范围; 全局搜索阶段采用 first-improvement 准则. 更重要的是, 本文分析并推导出启发式算法基本操作的目标增量性质, 提出目标增量法仅仅计算新解的目标函数增量而不是计算该解的整个目标函数值来降低算法的时间复杂度, 大大节省了 FIG 的计算时间.

2 问题描述

无等待流水调度问题(简称 NWFP)可以描述为 n 个任务 $\{J_1, J_2, \dots, J_n\}$ 连续在 m 台机器 $\{M_1, M_2, \dots, M_m\}$ 上以相同的顺序加工, 所有任务的工艺路线相同, 每台机器上任务加工的顺序也完全相同, 任务一旦开始加工不允许等待, 即等待只可能发生在第一台机器上. 具体约束可描述为: (1) 任务在机器上的加工顺序相同且确定; (2) 每道工序的加工时间预先给定; (3) 每台机器同时只能加工一个任务, 一个任务一次只能在一台机器上加工; (4) 任务不允许等待, 即要求任务一旦开始加工就必须连续进行直到加工完成.

该问题的任何调度都是任务集合 $\{J_1, J_2, \dots, J_n\}$ 元素的一个全排列. 为便于描述, 引入虚任务 $\pi_{[0]}$ 作为一个调度的初始任务, 它在各台机器上加工时间均为 0. 因此, 一个调度 π 可表示为 $(\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$, 其中 $\pi_{[i]} \in \{J_1, J_2, \dots, J_n\}$ ($1 \leq i \leq n$) 表示 π 中第 i 个位置上的任务. 设 $t_{k,i}$ 为任务 J_i ($i = 0, 1, \dots, n$) 在机器 M_k ($k = 1, 2, \dots, m$) 上的加工时间, $D_{i,j}$ 为任务 J_i 与 J_j 相邻时二者在最后一台机器

上的完工时间距离,根据文献[4]有

$$D_{i,j} = \max_{k=1,\dots,m} \left\{ \sum_{h=k}^m (t_{h,j} - t_{h,i}) + t_{k,i} \right\} \quad (1)$$

显然 $D_{0,j} = \sum_{h=1}^m t_{h,j} (j=1,\dots,n)$, 容易证明 $D_{i,j} > 0$.

记任务对 $(\pi_{[i]}, \pi_{[j]}) (0 \leq i, j \leq n \text{ 且 } i \neq j)$ 的完工时间距离为 $D_{[i],[j]}^\pi$, 调度 π 中 $\pi_{[i]} (0 \leq i < n)$ 及其直接后继 $\pi_{[i+1]}$ 间的完工时间距离记为 $d_{[i]}^\pi$, 则调度 π 的 total flowtime 可表示为

$$F(\pi) = \sum_{i=0}^{n-1} (n-i) D_{[i],[i+1]}^\pi = \sum_{i=0}^{n-1} (n-i) d_{[i]}^\pi \quad (2)$$

由此可见, 解决 $F_m | nwt | \sum C_i$ 问题就是要在所有可能的任务序列集合 Π 中找到这样一个任务序列 π^* , 使得

$$F(\pi^*) \leq F(\pi) \quad \forall \pi \in \Pi.$$

从式(1)可看出, $D_{i,j}$ 仅仅取决于 $t_{k,i}$, 而 $t_{k,i}$ 是预先给定和确定的, 任何两个任务之间的距离 $D_{i,j}$ 不会随调度的变化而变化. 因此, 任务集合 $\{J_1, J_2, \dots, J_n\}$ 中任何两个任务(包括虚任务)之间的距离可以预先求出并存储在矩阵 DM 中. 由于 $\pi_{[0]}$ 不可能出现于任何一个任务之后, 故令 $D_{i,0} = \infty$, 并规定同一任务间的距离也为 ∞ , 即 $D_{i,i} = \infty$. 因此, 矩阵 DM 可表达为

$$DM = \begin{bmatrix} \infty & D_{0,1} & D_{0,2} & \cdots & D_{0,n} \\ \infty & \infty & D_{1,2} & \cdots & D_{1,n} \\ \infty & D_{2,1} & \infty & \cdots & D_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & D_{n,1} & D_{n,2} & \cdots & \infty \end{bmatrix}_{(n+1) \times (n+1)}.$$

由于式(1)中计算 $D_{i,j}$ 的时间复杂度为 $O(m)$, 所以计算矩阵 DM 的时间复杂度为 $O(mn^2)$; 直接从 DM 中获取 $d_{[i]}^\pi$, 式(2)计算 $F(\pi)$ 的时间复杂度可减少为 $O(n)$.

3 目标增量法

启发式算法通常开始于一个活动解, 通过一些基本操作产生邻域, 在邻域中搜索更接近最优解的新解替换该活动解, 迭代该过程直到满足一定的停止条件. 对于 $F_m | nwt | \sum C_i$ 问题而言, 邻域的产生方法基本上可归纳为任务插入、删除、移位和对换等 4 个基本操作.

由式(1)、(2)可知, 一个调度序列的目标函数值 (total flowtime) 等于相邻任务间距离 AD (Adjacent Distance) 的加权和, AD 的值仅仅与这两个任务工序的加工时间有关. 因此, 对调度序列的任何一个基本操作(插入、删除、移位或对换)只会引起目标函数值在操作位置上的增加或减少, 序列中其它位置上任务的 AD 保持不变, 我们称相应的目标函数值变化量为目标增量. 由此可见, 基本操作是导致调度序列目标函数值发生变化的根本原因. 新序列的目标函数值就等于活动解的目标函数值与目标增量的和, 如果仅仅计算新序列的目标增量和而不是按传统方法通过式(2)计算目标函数值, 可大大节省计算时间, 称这种方法为目标增量法. 对于一个 $n=6, m=3$ 的 NWFP 例子, 图 1 和图 2 分别给出将任务 J_2 移动到任务 J_6 之后以及任务 J_2 和任务 J_4 对换位置的过程.

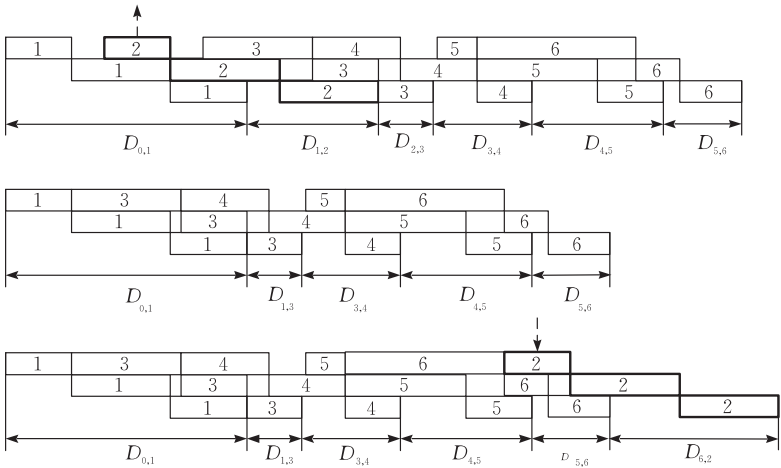


图 1 将任务 J_2 移动到任务 J_6 之后

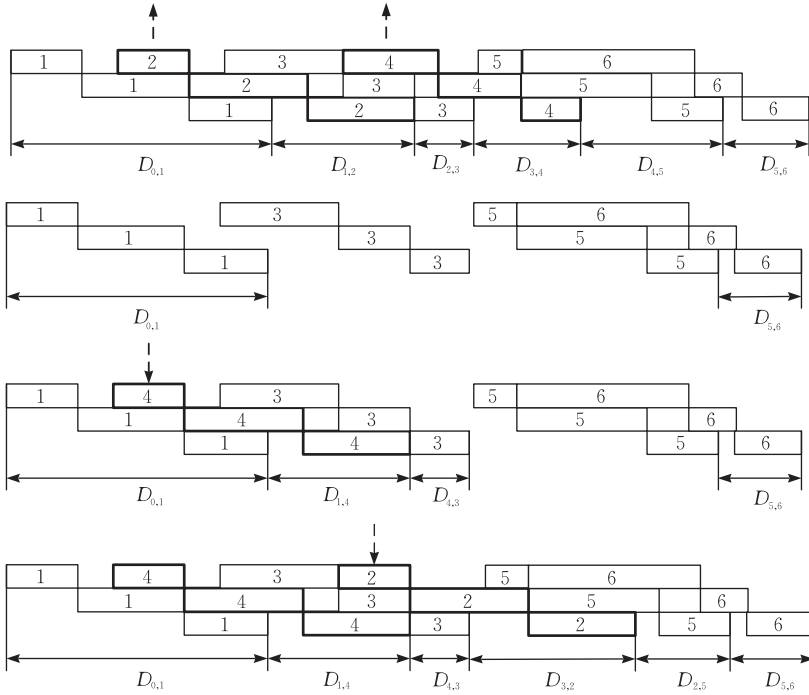
图 2 任务 J_2 与任务 J_4 对换位置

图 1 表明 π 中任务 J_2 的移位实际上是先将其从所在位置上删除,再插入到任务 J_6 之后的位置,将减少 $D_{1,2}$ 和 $D_{2,3}$ 、增加 $D_{1,3}$ 和 $D_{6,2}$,而 $D_{0,1}, D_{3,4}, D_{4,5}$ 和 $D_{5,6}$ 保持不变. 同样,在图 2 任务 J_2 和任务 J_4 的位置互换过程中,将减少 $D_{1,2}, D_{2,3}, D_{3,4}$ 和 $D_{4,5}$,增加 $D_{1,4}, D_{4,3}, D_{3,2}$ 和 $D_{2,5}$,而 $D_{0,1}$ 和 $D_{5,6}$ 保持不变.

设 π' 是某个操作作用于 π 后产生的一个邻居解,则目标函数值 $F(\pi') = F(\pi) + \Delta(\pi)$ (如果 π' 比 π 好,则增量为负值). 不同基本操作具有不同的目标增量性质,为了简化描述,设

$$x_{i,j}(\pi) = \begin{cases} 0, & i = n \\ (n-i)(D_{[i],[j]}^{\pi} - d_{[i]}^{\pi}), & \text{否则} \end{cases},$$

$$y_{i,j}(\pi) = \begin{cases} 0, & j = n \\ (n-j)(D_{[i],[j+1]}^{\pi} - d_{[j]}^{\pi}), & \text{否则} \end{cases},$$

$$s(k) = \begin{cases} 0, & k < 0 \\ \sum_{i=0}^k d_{[i]}^{\pi}, & k \geq 0 \end{cases}.$$

定理 1. 若将任务 J_q 插入到一个 $F_m | nwt | \sum C_i$ 序列 $\pi = (\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$ 中任务 $\pi_{[k]} (0 \leq k \leq n)$ 之后,则 total flowtime 将增加 $\Delta(n, k) = s(k-1) + (n-k+1)D_{[k],q}^{\pi} + y_{q,k}(\pi)$.

证明. (1) 当 $k=0$ (即任务 J_q 插入到 π 中虚任务 $\pi_{[0]}$ 之后成为第一个实任务) 时,形成的新序列为 $\pi' =$

$(\pi_{[0]}, J_q, \pi_{[1]}, \dots, \pi_{[n]})$, 由式 (2) 可知其 total flowtime 为

$$\begin{aligned} F(\pi') &= (n+1)D_{[0],q}^{\pi} + nD_{q,[1]}^{\pi} + \sum_{i=1}^{n-1} (n-i)d_{[i]}^{\pi} \\ &= F(\pi) + (n+1)D_{0,q} + nD_{q,[1]}^{\pi} - nd_{[0]}^{\pi} \\ &= F(\pi) + (n+1)D_{0,q} + y_{q,0}(\pi) \\ &= F(\pi) + s(-1) + (n+1)D_{0,q} + y_{q,0}(\pi) \\ &= F(\pi) + \Delta(n, 0). \end{aligned}$$

(2) 当 $1 \leq k \leq n-1$ 时,形成的新序列为 $\pi' = (\pi_{[0]}, \dots, \pi_{[k]}, J_q, \pi_{[k+1]}, \dots, \pi_{[n]})$, 由式 (2) 可知其 total flowtime 为

$$\begin{aligned} F(\pi') &= \sum_{i=0}^{k-1} (n+1-i)d_{[i]}^{\pi} + (n-k+1)D_{[k],q}^{\pi} + \\ &\quad (n-k)D_{q,[k+1]}^{\pi} + \sum_{i=k+1}^{n-1} (n-i)d_{[i]}^{\pi} \\ &= F(\pi) + \sum_{i=0}^{k-1} d_{[i]}^{\pi} + (n-k+1)D_{[k],q}^{\pi} + \\ &\quad (n-k)D_{q,[k+1]}^{\pi} - (n-k)d_{[k]}^{\pi} \\ &= F(\pi) + s(k-1) + (n-k+1)D_{[k],q}^{\pi} + y_{q,k}(\pi) \\ &= F(\pi) + \Delta(n, k). \end{aligned}$$

(3) 当 $k=n$ (即任务 J_q 插入到序列 π 的末端) 时,形成的新序列为 $\pi' = (\pi_{[0]}, \dots, \pi_{[n]}, J_q)$, 由 $y_{i,j}(\pi)$ 知 $y_{q,n}(\pi) = 0$, 所以根据式 (2) 可知其 total flowtime 为

$$F(\pi') = \sum_{i=0}^{n-1} (n+1-i)d_{[i]}^{\pi} + D_{[n],q}^{\pi}$$

$$= F(\pi) + \sum_{i=0}^{n-1} d_{[i]}^{\pi} + D_{[n],q}^{\pi} + y_{q,n}(\pi)$$

$$= F(\pi) + \Delta(n, n) \quad \text{证毕.}$$

以下定理和推论的证明类似于定理 1.

定理 2. 若删除 $F_m | n\tau wt | \sum C_i$ 序列 $\pi = (\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$ 中任务 $\pi_{[k]} (1 \leq k \leq n)$, 则 total flowtime 将减少 $\partial(n, k) = s(k-1) + (n-k)d_{[k-1]}^{\pi} - y_{\pi_{[k-1]},k}(\pi)$.

推论 1. 若将 $F_m | n\tau wt | \sum C_i$ 序列 $\pi = (\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$ 中位置 k_1 上的任务 $\pi_{[k_1]}$ 移位到位置 $k_2 (1 \leq k_1, k_2 \leq n)$ 上, 则 total flowtime 将增加 $\Delta(n-1, k_2-1) - \partial(n, k_1)$.

推论 2. 若将 $F_m | n\tau wt | \sum C_i$ 序列 $\pi = (\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$ 中的任务 $\pi_{[i]}$ 与 $\pi_{[j]}$ 位置对换 $(1 \leq i < j \leq n)$, 则 total flowtime 将增加 $\omega(i, j) = \begin{cases} x_{i-1,j}(\pi) + y_{\pi_{[j]},j}(\pi) + (n-i)(D_{[j],[i]}^{\pi} - d_{[i]}^{\pi}), & j=i+1 \\ x_{i-1,j}(\pi) + y_{\pi_{[j]},i}(\pi) + x_{j-1,i}(\pi) + y_{\pi_{[i]},j}(\pi), & \text{否则} \end{cases}$.

由于启发式算法在搜索解的过程中, 需要不断通过上述基本操作产生新解并评价这些新解(计算目标函数值), 而算法的主要时间开销在于评价这些新解上. 从定理 1 和定理 2 可以看出: 对于 NWFP 问题, 插入和删除操作仅作用于很少的变化点, 引起的目标增量只需要计算少数几个 AD 值的和, 时间复杂度为 $O(1)$, 而计算整个目标函数值的时间开销为 $O(n)$, 所以在启发式算法中采用目标增量法可以将其时间复杂度降低 1 阶. 同理, 由推论 1 和推论 2 可知, 仅计算移位和对换引起的目标增量也可将算法时间复杂度降低 1 阶.

根据上述基本操作的目标增量性质, 可构造任务最佳点插入(BIP)和任务对最佳对换(BSP)这两个基本目标增量方法. BIP 是寻找任务 $\pi_{[i]}$ 从位置 i 移除后插入到另一个位置 j (即任务 $\pi_{[j-1]}$ 之后)使得目标增量最小, 具体描述如下.

1. 根据定理 2 计算 $\partial(n, i) \leftarrow s(i-1) + (n-i)d_{[i-1]}^{\pi} - y_{\pi_{[i-1]},i}(\pi)$;
2. 将任务 $\pi_{[i]}$ 从 π 中删除形成任务序列 τ ;
3. For $k=0$ to $n-2$

根据定理 1 计算 $\Delta(n-1, k) \leftarrow s(k) +$

$$(n-k)D_{[\tau[k]],\pi_{[i]}}^{\tau} + y_{\pi_{[i]},k}(\tau) - d_{[k]}^{\tau};$$

4. $\Delta(n-1, n-1) \leftarrow s(n-2) + D_{\tau_{[n-1]},\pi_{[i]}}^{\tau}$;
5. $\mu(p) \leftarrow \min_{k=0,1,\dots,n-1} \{\Delta(n-1, k) - \partial(n, i)\}$;
6. 如果 $\mu(p) < 0$, 则从 π 中删除任务 $\pi_{[i]}$ 并插入到任务

$\tau_{[p]}$ 之后形成新序列 π , $F(\pi) \leftarrow F(\pi) + \mu(p)$;

7. 返回 π , 算法结束.

算法步 3 的时间复杂度为 $O(n)$, 步 4 为 $O(1)$, 故 BIP 的时间复杂度为 $O(n)$. BSP 依据推论 2 计算序列 $(\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$ 中不同位置上两个任务对换引起的目标增量, 在所有 $n(n-1)/2$ 个任务对换增量中, 如果最小增量小于 0, 则表明相应的位置为当前最佳对换位置, 对换相应任务对. 该过程类似于 pair-wise exchange^[20], 二者的区别在于前者只计算目标增量而后者是计算整个目标函数值, 前者的时间复杂度为 $O(n^2)$, 后者为 $O(n^3)$.

4 快速迭代贪婪算法

快速迭代贪婪算法 FIG 的基本思想如下: 根据问题的特点构造初始解, 并置为当前解; 对当前解依次采用分段式重构策略和迭代全局搜索进行局部寻优和全局寻优, 从产生的邻域中选择较好的新解取代当前解; 采用类模拟退火的接收准则判断新解是否取代当前解; 迭代上述过程直至满足停止条件; 最后将得到的优良解采用任务对最佳对换作进一步改进.

4.1 初始解生成算法

由式(2)知, $F(\pi)$ 是由序列 $(\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$ 中任务 $\pi_{[i]}$ 与任务 $\pi_{[i+1]}$ 之间距离 $d_{[i]}^{\pi} (0 \leq i \leq n-1)$ 的加权和决定的, 其中的权值 $n-i$ 是 i 的反函数. 由于越靠近调度序列前面其权越大, 为保证 total flowtime 越小, 应选择越小的 $d_{[i]}^{\pi}$, 其实质是确定一条从虚任务 $\pi_{[0]}$ 出发的最短 Hamilton 路. 为保证距离矩阵 DM 不被破坏, 预先将 DM 保存在 DM' 中, 即 $DM' \leftarrow DM$. 初始解产生法 ISG 通过对矩阵 DM' 操作产生一条最短 Hamilton 路, 其基本思想为: 设 $\mu_1 \leftarrow \emptyset, \mu_2 \leftarrow \{all\ jobs\}, i \leftarrow 1$; 为防止产生环路, 令 $D_{j,[i-1]}^{\pi} \leftarrow \infty (0 \leq j \leq n)$, 在矩阵 DM' 中第 $\pi_{[i-1]}$ 行寻找 $\min_{j=1,\dots,n} \{D_{[i-1],j}^{\pi}\}$ 对应的任务作为 $\pi_{[i]}, \mu_1 \leftarrow \mu_1 + \{\pi_{[i]}\}, \mu_2 \leftarrow \mu_2 - \{\pi_{[i]}\}, i \leftarrow i+1$, 重复上述过程直至 $\mu_2 = \emptyset$ 为止. 确定一条路径 $(\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]})$ 即为初始解 π_0 , 恢复 DM 的值即 $DM \leftarrow DM'$. ISG 算法具体描述如下.

算法 1. ISG.

1. $DM' \leftarrow DM$;
2. $\mu_1 \leftarrow \emptyset, \mu_2 \leftarrow \{all\ jobs\}, i \leftarrow 1$;
3. While $(\mu_2 \neq \emptyset)$

3. 1. For $j=0$ to n

$$D_{j,[i-1]}^{\pi} \leftarrow -\infty;$$

3. 2. $D_{[i-1],[i]}^{\pi} \leftarrow \min_{j=1,\dots,n} \{D_{[i-1],j}^{\pi}\}, \mu_1 \leftarrow \mu_1 + \{\pi_{[i]}\},$
 $\mu_2 \leftarrow \mu_2 - \{\pi_{[i]}\}, i \leftarrow i+1;$

4. $DM \leftarrow DM', \pi_0 \leftarrow (\pi_{[0]}, \pi_{[1]}, \dots, \pi_{[n]}), F(\pi_0) \leftarrow$
 $\sum_{i=0}^{n-1} (n-i)d_{[i]}^{\pi};$

5. 返回 π_0 , 算法结束.

算法步 1 的时间复杂度为 $O(mn^2)$, 步 3 为 $O(n^2)$, 故上述初始解产生算法的时间复杂度为 $O(mn^2)$.

4.2 分段式重构策略

分段式重构策略 SRS (Segment based Reconstruction Strategy) 是一个局部迭代搜索的过程, 通过随机分划子序列和逐段邻域搜索, 扩大寻优范围, 从而提高求解质量. 其步骤如下: 首先将调度序列 π 用 bN 个断点随机分成 $bN+1$ 段 (bN 为预先设置的常数, 通常很小), 第 i ($i=0, 1, \dots, bN$) 个序列段的首尾分别用首指针 $start_i$ 和尾指针 end_i 来指示. 对每个序列段进行 h 次 (h 等于该段段长或与之成比例) 如下分解与重构操作: 从该段中随机抽取 d 个任务 (d 是段长的 P_d 倍), 将这 d 个任务按照其原来在 π 中位置的倒置形成序列 π_d , 该段中剩下的任务按照原来的顺序形成待重构的部分序列 π_r^i ; 从 π_d 中逐一取出任务对序列 π_r^i 进行 BIP 操作, 重构出使该段更优的子序列. SRS 算法具体描述如下.

算法 2. SRS(π).

1. 将序列 π 随机分成 $bN+1$ 段, 第 i 段记作 $[start_i, end_i], i=0, 1, \dots, bN;$

2. For $i=0$ to bN

2. 1. $h \leftarrow [start_i, end_i]$ 中任务个数, $d \leftarrow h \cdot P_d;$

2. 2. For $j=1$ to h

2. 2. 1. $\pi' \leftarrow \pi, \pi_d \leftarrow \emptyset;$

2. 2. 2. For $k=0$ to $d-1$

从 π' 的段 $[start_i, end_i]$ 中随机取出一个任务插入 π_d ;

2. 2. 3. 将 π_d 中的任务按其在 π' 中位置的降序排序,
 $[start_i, end_i]$ 剩余任务构成待重构序列 π_r^i ;

2. 2. 4. For $k=0$ to $d-1$

$\pi' \leftarrow$ 将任务 $\pi_{d[k]}$ 对 π_r^i 进行 BIP 插入操作;

2. 2. 5. 如果 $F(\pi') < F(\pi)$, 则 $\pi \leftarrow \pi';$

3. 返回 π , 算法结束.

由上可知, SRS 算法的时间复杂度集中在步 2, 其中步 2.2 的时间复杂度为 $O(n^3)$, 而 bN 为较小常数, 故 SRS 的时间复杂度为 $O(n^3)$.

对于 $n=10, m=4$ 调度问题, 图 3 示例了调度实例 $\pi = (J_2, J_1, J_3, J_5, J_6, J_4, J_7, J_8, J_9, J_{10})$ 的单次分解与重构过程.

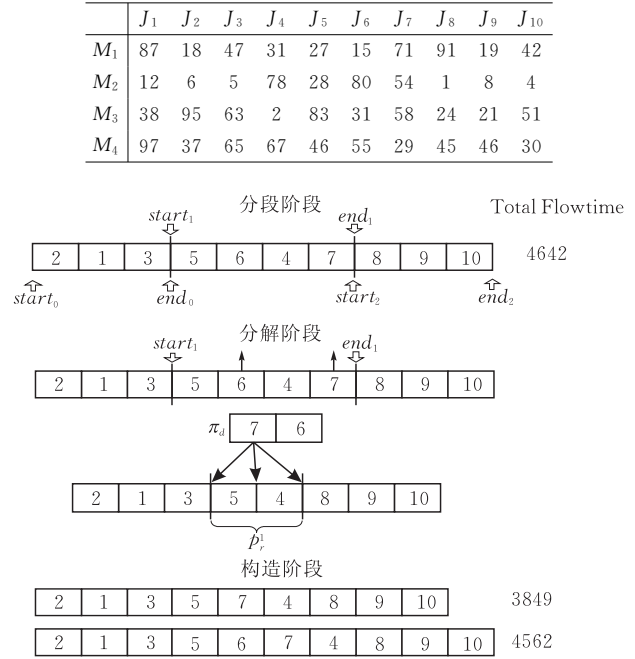


图 3 单次分解与重构过程

为便于描述, 令数组 $(2, 1, 3, 5, 6, 4, 7, 8, 9, 10)$ 代表序列 π . 初始时 π 的目标函数值为 4642; 两个断点 (短竖线) 将 π 分割成长度分别为 3, 4, 3 的三段, 分别为 $(2, 1, 3)$, $(5, 6, 4, 7)$ 和 $(8, 9, 10)$; 取 $P_d = 0.7$, 中段 $(5, 6, 4, 7)$ 的分解重构过程如下: 从该段中随机抽取 $\lfloor 4 \times 0.7 \rfloor = 2$ 个任务 J_6 和 J_7 , 按照其原来在 π 中的倒置顺序形成序列 $\pi_d = (7, 6)$, 该段中剩下的任务则构成待重构的部分序列 $\pi_r^i = (5, 4)$; 从 π_d 中依次取出任务对 π_r^i 进行 BIP 插入直至 π_d 为空, 重构后的新子序列依次为 $(5, 7, 4)$ (目标函数值为 3849) 和 $(5, 6, 7, 4)$ (目标函数值为 4562), 而 π 经过该段重构后得到的新序列为 $(2, 1, 3, 5, 6, 7, 4, 8, 9, 10)$, 对应目标函数值 4562.

表 1 给出了 SRS 算法的完整分段重构过程: 初始序列 $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ 的目标函数值为 5172, 被分割成长度分别为 3, 4, 4 三段; 对第 0 段进行迭代分解与重构后的最优解为 $(2, 1, 3, 4, 5, 6, 7, 8, 9, 10)$, 对应目标函数值 4733; 对第 1 段进行迭代分解与重构后的最优解为 $(2, 1, 3, 5, 6, 7, 4, 8, 9, 10)$, 对应目标函数值 4562; 对第 2 段进行迭代分解与重构后的最优解为 $(2, 1, 3, 5, 6, 7, 4, 10, 9, 8)$, 对应目标函数值 4551, 即为最终结果.

表 1 SRS 算法的实例执行过程

第 i 段	迭代	当前序列	π_d	π_r^i	待插任务	子序列	目标函数值
初始序列		(1,2,3 4,5,6,7 8,9,10)					5172
$i=0$	1	(1,2,3 4,5,6,7 8,9,10)	(3,2)	(1)	J_3	(3,1 4,5,6,7 8,9,10)	4199
				(1)	J_2	(3,1,2 4,5,6,7 8,9,10)	4776
	2	(3,1,2 4,5,6,7 8,9,10)	(2,1)	(3)	J_2	(2,3, 4,5,6,7 8,9,10)	3903
				(3)	J_1	(2,1,3 4,5,6,7 8,9,10)	4733
				(3)	J_1	(3,1 4,5,6,7 8,9,10)	4199
				(3)	J_2	(3,1,2 4,5,6,7 8,9,10)	4776
$i=1$	1	(2,1,3 4,5,6,7 8,9,10)	(7,6)	(4,5)	J_7	(2,1,3 4,5,7 8,9,10)	3984
				(4,5)	J_6	(2,1,3 6,4,5,7 8,9,10)	4688
	2	(2,1,3 6,4,5,7 8,9,10)	(5,6)	(4,7)	J_5	(2,1,3 5,4,7 8,9,10)	3929
				(4,7)	J_6	(2,1,3 5,6,4,7 8,9,10)	4642
	3	(2,1,3 5,6,4,7 8,9,10)	(7,6)	(5,4)	J_7	(2,1,3 5,7,4 8,9,10)	3849
				(5,4)	J_6	(2,1,3 5,6,7,4 8,9,10)	4562
	4	(2,1,3 5,6,7,4 8,9,10)	(4,7)	(5,6)	J_4	(2,1,3 5,6,4 8,9,10)	3849
				(5,6)	J_7	(2,1,3 5,6,7,4 8,9,10)	4562
$i=2$	1	(2,1,3 5,6,7,4 8,9,10)	(10,9)	(8)	J_{10}	(2,1,3 5,6,7,4 10,8)	3859
				(8)	J_9	(2,1,3 5,6,7,4 10,9,8)	4551
	2	(2,1,3 5,6,7,4 10,9,8)	(8,9)	(10)	J_8	(2,1,3 5,6,7,4 10,8)	3859
				(10)	J_9	(2,1,3 5,6,7,4 10,9,8)	4551
				(10)	J_8	(2,1,3 5,6,7,4 10,8)	3859
				(10)	J_9	(2,1,3 5,6,7,4 10,9,8)	4551

4.3 迭代全局搜索

迭代全局搜索法 IGS(Iterated Global Search) 采用循环删除-插入操作,对分段式重构后产生的解作进一步改进.IGS 算法的迭代过程如下:随机不重复地从当前序列 $\pi=(\pi_{[0]},\pi_{[1]},\cdots,\pi_{[n]})$ 中抽取出一个实任务 $\pi_{[k]}(1\leq k\leq n)$,依次计算 $\pi_{[k]}$ 插入 π 中剩余序列可能位置的目标增量(不可插入虚任务 $\pi_{[0]}$ 之前),选择第一个移位目标增量小于 0 的插入点,将 $\pi_{[k]}$ 插入该点得到的新解作为当前解.将 π 中所有实任务都做上述操作,若当前解在这 n 次循环中有至少 1 次被改进,则继续下一次迭代;否则停止,算法结束.IGS 的单次搜索过程类似 BIP,不同之处在于:前者中邻居解的接收采用 first-improvement 准则,即接收领域中搜索到的第一个有改进的解;而后者采用 best-improvement 准则,即接收领域中搜索到的最优改进解.IGS 算法具体描述如下.

算法 3. IGS(π).

1. $flag\leftarrow true$;
2. While ($flag=true$)

2.1. $flag\leftarrow false$;

2.2. For $i=1$ to n

2.2.1. 从 π 中随机且不重复地取出一个实任务 $\pi_{[k]}$;

2.2.2. For $j=1$ to n

2.2.2.1. 根据推论 1, 计算插入点 j 对应的移位目标增量 $\Delta(n-1,j-1)-\partial(n,k)$;

2.2.2.2. 如果 $\Delta(n-1,j-1)-\partial(n,k)<0$, 则将任务 $\pi_{[k]}$ 插入第 $j-1$ 个任务之后, 得到 π^* , $\pi\leftarrow\pi^*$, $flag\leftarrow true$;

3. 返回 π ,算法停止.

算法 IGS 的时间复杂度集中在步 2,步 2.2 的时间复杂度为 $O(n^2)$;由于 $flag$ 指示的停止条件不定,所以 IGS 的时间复杂度需根据外循环次数来确定.

4.4 FIG 算法描述

快速迭代贪婪算法 FIG 由初始解产生、分段式重构和迭代全局搜索 3 个阶段组成:调用 ISG 算法构造初始解并赋值给当前解;将当前解随机分成若干段,采用分段式重构策略 SRS 进行局部优化和搜索;对改进后的解采用迭代全局搜索 IGS 和任务对最佳对换法 BSP,进一步提高解的质量.其中邻域搜索过程均基于基本目标增量法 BIP 和 BSP,大大减少了时间开销.

分段式重构和迭代全局搜索后得到的解序列可能比已经得到的最好解差,是否进一步迭代搜索需要根据一定的准则来判断.由于模拟退火算法(SA)可在保证收敛的情况下以一定概率跳出局部最优,故本文采用一种类模拟退火的接收准则,将 π 改进后的解 π' 以概率 $\exp\{-(F_n(\pi')-F_n(\pi))/T\}$ 接收并替换当前活动解,式中 T 为控制参数,随着算法的运行逐渐减小. T 的值根据具体实例确定并初始化为 $T_0=t\sum_{i=1}^n\sum_{k=1}^mt_{k,i}/(10nm)$,退火过程中的冷却机制采用文献[21]中的 $T_{s+1}=\lambda T_s(0<\lambda<1)$ 以维持计算速率与达到低能态二者之间的平衡.FIG 算法具体描述如下.

算法 4. FIG.

1. 关键参数设置： bN, P_d, t, λ ;
2. 调用 ISG 算法产生初始解 π_0 ;
3. $\pi \leftarrow \pi_0, \pi_b \leftarrow \pi$;
4. While (不满足终止条件)

4. 1. $\pi \leftarrow \text{SRS}(\pi)$;

4. 2. $\pi^* \leftarrow \text{IGS}(\pi)$;

4. 3. 如果 $F(\pi^*) < F(\pi)$, 则 $\pi \leftarrow \pi^*$;

4. 3. 1. 如果 $F(\pi) < F(\pi_b)$, 则 $\pi_b \leftarrow \pi$;
否则

4. 3. 2. 如果 $\text{random} \leq \exp\{-(F(\pi^*) - F(\pi))/T\}$,
则 $\pi \leftarrow \pi^*, T \leftarrow \lambda T$;
5. $\pi_b \leftarrow \text{BSP}(\pi_b)$;
6. 返回 π_b , 算法结束.

FIG 算法的时间复杂度集中在第 2、4、5 步,其中步 2 为 $O(mn^2)$;步 5 在不采用目标增量法时为 $O(n^3)$,采用时为 $O(n^2)$;尽管步 4 的时间复杂度需根据 While 循环次数确定,但对于主要步 4.1 和 4.2,在不采用目标增量法时均为 $O(n^4)$,采用时为 $O(n^3)$;由此可见,采用目标增量法的 FIG 的时间复杂度比不采用目标增量法可降低 1 阶.

5 模拟实验结果及性能分析

为分析本文所提出的迭代贪婪算法的性能,下面将 FIG 算法与目前较有效的复合启发式算法 PH1p^[16]、元启发式算法 SRTS^[14] 和 DPSOvnd^[18] 进行比较. SRTS 和 DPSOvnd 将 Taillard^[15] 的 110 个 Benchmark 实例分成 4 组进行比较,为给出更详尽的比较结果,本文按照相同的分组方式将 110 个实例共分成 11 组,分别记为 T1~T11,即每 10 个实例 1 组,如 T1 表示 ta001~ta010. FIG 算法主要参数设置如下:当 n 为 20、50、100 和 200 时, bN 分别取 4、5、5、10, $P_d=0.7, t=0.37, \lambda=0.99$, 停止条件为最大迭代次数 250 次. 所有算法均用 Visual C++ 实现,并在 Pentium 2.93GHz、512MB RAM PC 机上执行.

分别从 ARPD^[4](单位:%)和所需 CPU 计算时间(单位:s)两方面对以上算法的性能和效率进行比较. 每组实例的 ARPD(其中 $N=10, B_i$ 为 4 个算法求解实例 i 的最小 total flowtime)结果如表 2 所示.

表 2 算法的性能和效率比较

Group	n	m	SRTS		PH1p		DPSOvnd		FIG	
			ARPD	CPU	ARPD	CPU	ARPD	CPU	ARPD	CPU
T1	20	5	0.151	1.553	1.709	0.091	0.000	0.863	0.000	0.420
T2	20	10	0.000	1.506	2.229	0.151	0.000	0.837	0.000	0.420
T3	20	20	0.000	1.512	1.755	0.392	0.000	0.840	0.000	0.420
T4	50	5	0.323	13.854	4.074	0.746	0.203	7.696	0.000	4.615
T5	50	10	0.482	14.303	3.834	2.441	0.176	7.946	0.000	4.699
T6	50	20	0.567	14.061	3.035	6.768	0.199	7.812	0.008	4.657
T7	100	5	0.970	103.161	4.469	6.137	0.072	57.312	0.268	47.957
T8	100	10	0.510	108.856	4.770	19.971	0.102	60.476	0.183	46.866
T9	100	20	0.404	107.050	4.266	50.493	0.122	59.472	0.041	46.908
T10	200	10	1.812	996.445	4.673	194.685	0.009	553.580	0.445	302.509
T11	200	20	1.042	1003.422	4.503	504.495	0.078	557.457	0.612	303.642
平均			0.569		3.574		0.087		0.142	

从表 2 可以看出:在性能方面,PH1p 算法随着问题规模的扩大呈上升趋势且波动较大,对所有问题其 ARPD 都比其它 3 个算法大得多,最大值在问题 T8 达到 4.77%,最小值也超过 1%,11 组实例的平均 ARPD 为 3.574%. SRTS 算法的平均 ARPD 为 0.569%,仅优于 PH1p,在问题 T2 和 T3 时为最低值 0,在问题 T10 和 T11 时则分别达到峰值 1.812%和 1.042%,其变化趋势类似于 PH1p. FIG 的平均 ARPD 为 0.142%,很接近于 DPSO vnd 的平均 ARPD(0.087%),二者差值约为 0.05%;当问题规模较小时(T1~T5),FIG 的 ARPD 始终为 0,且等于或小于 DPSOvnd 的 ARPD,二者差值在

0.2%左右;随着问题规模的扩大,除特殊点 T9 之外,FIG 的 ARPD 相对 DPSOvnd 的 ARPD 较大,且二者差值呈增长趋势.

在效率方面,PH1p 的效率最高,FIG 其次, SRTS 的效率最差. 例如,就规模为 200×10 的问题 T10 而言,PH1p 需要的时间为 195s,FIG 需要 303s 的时间,DPSOvnd 需要 554s 的时间,而 SRTS 需要 996s 的时间. 对于所有问题而言,FIG 的 CPU 时间约为 DPSOvnd 所需时间的 1/2,为 SRTS 所需时间的 1/3;FIG 的 CPU 时间在问题规模较小时接近于 PH1p 所需时间,并随着问题规模的扩大而增加,但 FIG 的性能远远好于 PH1p.

综上所述,对于所有的任务机器的组合进行的测试,FIG 算法在性能方面优于 PH1p 算法和 SRTS 算法,略逊于 DPSOvnd 算法,在效率方面优于 SRTS 算法和 DPSOvnd 算法,比 PH1p 算法略差。

6 结 论

本文讨论最小化总完工时间的无等待流水调度问题,根据问题的特点分析出调度序列中任务间的时间参数具有相对独立性,并证明出算法基本运算的目标增量性质;根据这些性质提出一种快速迭代贪婪算法 FIG,构造出启发式算法 ISG 产生初始解,建立了用于进一步提高解质量的分段式重构策略和迭代全局搜索算法,采用类似于模拟退火机制的邻域接收准则以防止算法陷入局部最优.整个解的搜索过程基于两种目标增量方法 BIP 和 BSP,快速评估由插入、删除、移位或对换操作对一个解产生邻域.通过 110 个 Benchmark 实例将提出 FIG 算法与当前较好的启发式算法 PH1p 和元启发式算法 SRTS、DPSOvnd 进行比较,实验结果表明 FIG 在性能上优于 SRTS 和 PH1p,略逊于 DPSOvnd,在效率上优于 SRTS 和 DPSOvnd,略逊于 PH1p.因此,FIG 算法适用于求解大规模无等待流水调度问题。

参 考 文 献

- [1] Pinedo M. Scheduling: Theory, Algorithm, and Systems. Englewood Chills, NJ: Prentice-Hall, 1995
- [2] Hall N G, Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research, 1996, 44(3): 510-525
- [3] Rajendran C. A no-wait flowshop scheduling heuristic to minimize makespan. Journal of the Operational Research Society, 1994, 45(4): 472-478
- [4] Li Xiaoping. Heuristics for large scale flowshop problems [Postdoctoral Research Report]. Beijing: Tsinghua University, 2004(in Chinese)
(李小平. 启发式求解几个大规模流水调度问题[博士后研究报告]. 北京: 清华大学, 2004)
- [5] MacCarthy B L, Liu J. Addressing the gap in scheduling research: A review of optimization and heuristic methods in production scheduling. International Journal of Production Research, 1993, 31(1): 59-79
- [6] Van Deman J M, Baker K R. Minimizing mean flow time in the flowshop with no intermediate queues. AIIE Transactions, 1974, 6(1): 28-34
- [7] Adiri I, Pohoryles D. Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times. Naval Research Logistics Quarterly, 1982, 29(3): 495-504
- [8] Van der Veen J A A, Van Dal R. Solvable cases of the no-wait flowshop scheduling problem. Journal of the Operational Research Society, 1991, 42(11): 971-980
- [9] Rajendran C, Chaudhuri D. Heuristic algorithms for continuous flow-shop problem. Naval Research Logistics, 1990, 37(5): 695-705
- [10] Bonney M C, Gundry S W. Solutions to the constrained flow-shop sequencing problem. Operational Research Quart, 1976, 27(4): 869-883
- [11] King J R, Spachis A S. Heuristics for flow-shop scheduling. International Journal of Production Research, 1980, 18(3): 343-357
- [12] Bertolissi E. Heuristic algorithm for scheduling in the no-wait flow-shop. Journal of Materials Processing Technology, 2000, 107(1-3): 459-465
- [13] Chen C L, Neppalli R V, Aljaber N. Generic algorithms applied to the continuous flow shop problem. Computers and Industrial Engineering, 1996, 30(4): 919-929
- [14] Fink A, Voß S. Solving the continuous flow-shop scheduling heuristic to minimize makespan. Journal of the Operational Research, 2003, 151(3): 400-414
- [15] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research, 1993, 64(2): 278-285
- [16] Aldowaisan T, Allahverdi A. New heuristics for m-machine no-wait flowshop to minimize total completion time. OMEGA, 2004, 32(5): 345-352
- [17] Kumar A, Prakash A, Shankar R, Tiwari M K. Psychological algorithm based approach to solve continuous flowshop scheduling problem. Expert Systems with Applications, 2006, 31(3): 504-514
- [18] Pan Q K, Tasgetiren M F, Liang Y C. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Computers and Operations Research, 2008, 35(9): 2807-2839
- [19] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 2007, 177(3): 2033-2049
- [20] Wang L, Zheng D Z. An effective hybrid optimization strategy for job-shop scheduling problem. Computers and Operations Research, 2001, 28(6): 585-596
- [21] Osman I H, Potts C N. Simulated annealing for permutation flow-shop scheduling. Omega, 1989, 17(6): 551-557



ZHU Xia, born in 1982, Ph. D. candidate. Her main research interests include scheduling optimization, service oriented computing.

LI Xiao-Ping, born in 1970, Ph. D. , associate professor, Ph. D. supervisor. His main research interests include scheduling optimization, service oriented computing, manufacturing interoperability.

WANG Qian, born in 1946, professor, Ph. D. supervisor. Her main research interests include information integration, database technology, CSCWD.

Background

The aim of this work is to propose an effective and efficient heuristic algorithm for total flowtime minimization in no-wait flowshop problems. This work is supported mainly by the National Natural Science Foundation of China under grant No. 60504029 called “objective increment based composite heuristics for large scale no-wait flowshops”. It focuses on no-wait flowshop problems (NWFPs for short) which are important kind of constrained flowshop problems. In NWFPs, the operations of each job have to be continuously processed. No interruption is permitted either on or between machines. NWFPs widely exist in metallurgy, plastic molding, chemical processing and food processing industries. Modern manufacturing systems such as JIT systems, flexible manufacturing environments and robotic cells can also be modeled as NWFPs. The optimization objective in NWFPs can be modeled as the weighted sum of “distance” of adjacent jobs in a sequence. The “distance” is related only to the processing times of the related adjacent jobs but independent on other jobs in the sequence. Hence, it can be predetermined. Whether a new generated schedule is better or not than the original one can be judged just by calculating the sum of all the

objective increments by the change of job positions, instead of calculating all time parameters of jobs in the whole schedule. Therefore, the time complexity of algorithms can be reduced. Objective increment properties are analyzed for fundamental operations of heuristics. Based on the properties, two fundamental methods are introduced for evaluating schedules fast. A fast iterated greedy algorithm FIG is proposed. In FIG, an initial solution generating method is constructed, and then a segment based reconstruction strategy and an iterative improvement procedure are developed for local and global search respectively to improve the current solution. In addition, an SA-like acceptance criterion is adopted to escape from local optimal. Experimental results show that, FIG outperforms PH1p and SRTS algorithms but it is little worse than DPSOvnd algorithm on effectiveness. On efficiency, FIG is better than SRTS and DPSOvnd but a little worse than PH1p. Therefore, FIG may be suitable for large scale NWFPs. This work is also supported by the National Natural Science Foundation of China under grant No. 60672092 and the National High Technology Research and Development Program (863 Program) of China under grant No. 2008AA04Z103.