文章编号：1001-0920(2009)08-1203-06

# 一种基于 PFSP 性质的深度优先搜索算法

李文超[1,2]，严洪森[1]

(1. 东南大学 a. 自动化学院，b. 教育部复杂工程系统测量与控制重点实验室，
南京 210096；2. 江苏大学 交通运输系，江苏 镇江 212013)

**摘 要**：三机以上同顺序 Flow-shop 问题(PFSP)是著名的 NP 完全问题. 在充分利用 PFSP 自身特性的基础上，提出一种可变路径的深度优先搜索算法. 该算法在搜索过程中根据需要采用两种不同邻域，在必要时将 PFSP 转化为一个指派问题，自动变更搜索路径，以避免陷入局部最优解. 数值仿真实验表明，该算法对于大规模 PFSP 能取得良好的计算结果.

**关键词**：同序 Flow-shop 问题；指派问题；深度优先搜索

**中图分类号**：TP301 **文献标识码**：A

# Depth-first search algorithm based on special properties of PFSP

*LI Wen-chao*[1,2], *YAN Hong-sen*[1]

(1a. School of Automation, 1b. Key Laboratory of Measurement and Control of CSE, Ministry of Education, Southeast University, Nanjing 210096, China; 2. Department of Transportation, Jiangsu University, Zhenjiang 212013, China. Correspondent: LI Wen-chao, E-mail: zeropoint@ujs.edu.cn)

**Abstract**: The permutation flow-shop problem (PFSP) with more than three machines is a famous NP-complete problem. A variable path depth-first search algorithm is proposed by making good use of the characteristic of PFSP. The algorithm searches in two different kinds of neighborhood and PFSP is transformed into an assignment problem when necessary to avoid being trapped in a local optimal solution. Numerical experiments show that the proposed algorithm has excellent performance for large-scale flow-shop problem.

**Key words**: Permutation flow-shop problem; Assignment problem; Depth-first search

## 1 引 言

作为典型的组合优化问题，以最小完工周期为目标的同顺序 Flow-shop 问题(PFSP)，多年来一直深受研究者的关注. 总体而言，解决该问题的方法大体可分为两类：一类是精确算法，这方面的文献不多，且多出现在问题提出的早期；另一类是近似算法，这方面的文献较多，尤其是 Garey 等证明该问题为 NP 完全问题以后[1]，对该问题的研究大多集中于近似算法.

精确算法的典型代表是分枝定界方法，如 Haouari 等提出的基于局部搜索的分枝定界方法[2]，越民义等提出的消去准则[3]和定界方法[4]，Carlier 等提出的定界方法[5]等. 这类方法的特点是利用问题本身的特性来估计其上界，排除一些可行解，直至得到最优解. 但该问题的解空间是以 $n!$ 方式增长的，故这类方法对于大规模情况并不适用.

近似算法以启发式算法为主，可分为以下 3 类：规则式、迭代式和元启发式算法. 规则式算法按某一规则或指标对所有工件排列得到较优序，其优点是计算量小，算法简单. 比较典型的有 CDS 算法[6]、NEH 算法[7]等. 迭代式算法通常是在规则式算法所得较优解的基础上，以某种方式产生新的可行解，比较优劣并反复迭代，直至满足某个条件为止. 如 Suliman 的基于 CDS 的算法[8]，Woo 等的基于工件特性的算法[9]等. 这些算法所得的解一般优于规则式所得的解，但计算复杂度相应增加. 元启发式算法及其改进算法得到普遍的应用. 常用的有模拟退火算法、禁忌搜索算法、遗传算法等. 如 Osman 等的简单模拟退火算法[10]，Ogbu 等的修正模拟退火算法[11]，Nowicki 等[12]，Grabowski 等[13]和金锋等[14]的不同禁忌搜索算法等. 利用遗传算法研究该问题

**收稿日期**：2008-07-30；**修回日期**：2008-10-28.

**基金项目**：国家 863 计划专题基金项目(2007AA04Z112)；国家自然科学基金项目(60574062,50875046).

**作者简介**：李文超(1974—)，男，河南新密人，讲师，博士生，从事知识化制造、人工智能的研究；严洪森(1957—)，男，浙江江山人，教授，博士生导师，从事生产计划与调度、知识化制造等研究.

的文献较多，如 Reeves 的基于不同遗传算子的算法[15,16]，Wang 等的混合遗传算法[17] 等. 这些由元启发式算法所得到的解大多优于前两种方法得到的解，尤其是当问题规模较大时，由这类算法得到的解明显优于前者. 但这类方法引入了随机性，也存在一些缺点，如对问题参数敏感、依赖初始解等不稳定特征.

本文在综合考虑 PFSP 自身特性的基础上，提出在寻优过程中采取变路径的思路，必要时改变候选解的生成邻域，并将搜索过程中得到较优子代特性记录下来，形成一个权矩阵，当陷入局部最优时，可用该矩阵将原问题转化为指派问题，从而得到新的解以跳离局部最优. 由于迭代过程相对较少，降低了算法的计算量.

## 2 问题描述及相关理论

### 2.1 问题描述

同顺序 Flow-shop 问题(PFSP) 是指：$n$ 个工件 $J_1,J_2,\cdots,J_n$ 在 $m$ 台机器上生产时，在满足以下生产要求的情况下：

1) 每个工件按相同次序通过 $m$ 台机器进行加工；

2) 每台机器上工件加工次序都相同；

3) 每个工件在每台机器上加工时间是确定的；

4) 每台机器在同一时间只能加工一个工件；

5) 每个工件同一时间只能在一台机器上加工.

确定各个工件在机器上的加工顺序，使得某些性能指标达到最优. 通常以工件最大完工期作为指标.

PFSP 的解(即工件在机器上的加工顺序) 可用排列 $\boldsymbol{\omega}=(\omega(1),\cdots,\omega(i),\cdots,\omega(n))$ 表示，其中 $\omega(i)(i=1,2,\cdots,n)$ 为排在第 $i$ 个位置上的工件. 记 $C_{\max}(\boldsymbol{\omega})$ 为序列 $\boldsymbol{\omega}$ 的最大完工期，$\Pi$ 为 PFSP 所有解的集合，则该问题的最优解为

$$C_{\max}(\boldsymbol{\omega}^*)=\min_{\boldsymbol{\omega}\in\Pi}(C_{\max}(\boldsymbol{\omega})). \tag{1}$$

记 $a_{i\omega}(j)$ 为序列 $\boldsymbol{\omega}$ 中第 $j$ 个位置工件 $\omega(j)$ 在第 $i$ 台机器加工所用时间，$n$ 个工件在 $m$ 台机器上的加工时间可用 $m\times n$ 矩阵 $A(\boldsymbol{\omega})$ 表示，即工时矩阵. 序列 $\boldsymbol{\omega}$ 的相应最大完工期可表示为[12]

$$C_{\max}(\boldsymbol{\omega})=\max_{1=l_0\leqslant l_1\leqslant\cdots\leqslant l_m=n}\sum_{i=1}^{m}\sum_{j=l_{i-1}}^{l_i}a_{i\omega(j)}. \tag{2}$$

### 2.2 块与邻域

将式(2) 中的求和元素 $a_{i\omega(j)}$ 用直线连接起来，可形成一条起始于 $a_{1\omega}(1)$ 终止于 $a_{m\omega(n)}$ 的折线，该折线称为可行线. 在序列 $\boldsymbol{\omega}$ 的工时矩阵 $A(\boldsymbol{\omega})$ 中，共有 $C_{n+m-2}^{m-1}$ 条可行线[18]. 可行线上所有元素之和称为该可行线的可行和. 显然，式(2) 中最大完工期为所有可行和中最大者，即最大可行和. 称其所在可行线为最大可行线[4](不少文献称为关键路线). 最大可行线上水平线段总数记为 $l$，第 $i$ 条水平线段上元素所在行记为 $m_i$，$1\leqslant m_i\leqslant m$，$1\leqslant i\leqslant l$. 则相应的竖直线段数为 $l-1$，$l$ 和 $l+1$(根据折线不同形态确定). 每条竖直折线对应一个工件，该工件称为关键工件；每条水平折线对应一台机器，该机器称为关键机器. 关键工件在序列 $\boldsymbol{\omega}$ 中所在位置按先后顺序可记为一个序列 $u_0,\cdots,u_l$，$u_0=1$，$u_l=n$. 第 $k$ 个水平线段上的元素可表示为 $a_{m_k\omega(j)}$，$1\leqslant k\leqslant l$，$u_{k-1}\leqslant j\leqslant u_k$. 由此可对块作如下定义：

**定义 1**[12] 所有含有元素 $a_{m_k\omega(j)}$($1\leqslant k\leqslant l$，$u_{k-1}\leqslant j\leqslant u_k$) 的工件所组成的有序集合($\omega(u_{k-1})$，$\cdots,\omega(u_j),\cdots,\omega(u_k)$) 组成第 $k$ 个块，记为 $B_k$.

显然，块 $B_k$ 由关键工件 $\omega(u_{k-1}),\omega(u_k)$ 及其内部工件所组成. 内部工件可定义为块内工件，表示为

$$B_{k\,\mathrm{in}}=B_k-\{\omega(u_{k-1}),\omega(u_k)\},\ 1\leqslant k\leqslant l. \tag{3}$$

对 PFSP 解 $\omega$ 中一些工件的位置进行调整，会得到一些新的解，这些新解的集合通常称为 $\omega$ 的一个邻域. 调整工件的操作方法有多种，已证明较有效的方法主要是插入和交换方法[19](不少文献使用插入方法). 二者所产生的邻域分别记为 $N_{\mathrm{ins}}(\boldsymbol{\omega})$ 和 $N_s(\boldsymbol{\omega})$. 下面分别对它们进行讨论.

#### 2.2.1 插入操作

记 $\mathrm{ins}=(p,q)(p,q\in(1,2,\cdots,n),p\neq q)$ 为排列 $\boldsymbol{\omega}$ 的一个插入操作，即将 $\boldsymbol{\omega}$ 中处于第 $p$ 个位置的工件取出，插入到第 $q$ 个位置上，得到一个新的解序列

$$\boldsymbol{\omega}_{\mathrm{ins}}=\begin{cases}(\omega(1),\cdots,\omega(p-1),\omega(p+1),\cdots,\omega(q),\\ \omega(p),\omega(q+1),\cdots,\omega(n)),p<q;\\ (\omega(1),\cdots,\omega(q-1),\omega(p),\omega(q),\omega(q+1),\\ \cdots,\omega(p-1),\omega(p+1),\cdots,\omega(n)),p>q.\end{cases}$$

由插入操作所形成 $\boldsymbol{\omega}$ 的邻域为

$$N_{\mathrm{ins}}(\boldsymbol{\omega})=\{\boldsymbol{\omega}_{\mathrm{ins}}\mid\mathrm{ins}\in I\}. \tag{4}$$

其中

$$I=\{(p,q)\mid q\neq p-1,p,q\in\{1,2,\cdots,n\}\}.$$

式中 $q\neq p-1$，以避免产生与操作 $(p-1,p)$ 相重复的解.

由 $N_{\mathrm{ins}}$ 的定义可知，该邻域有 $(n-1)^2$ 个元素. 对其中每个解的评估时间复杂度为 $O(mn)$[20]，则搜索评估整个邻域的时间复杂度为 $O(mn^3)$. 随着工件数目 $n$ 的增加，需要评估邻域的数目也将急剧增加. 为此，有的学者基于 PFSP 问题本身的性质，提出一些缩小邻域 $N_{\mathrm{ins}}$ 搜索范围的方法. 如 Nowicki[12] 和 Grabowski 等[13] 提出将序列 $\boldsymbol{\omega}$ 块中元素插入到相

邻块中，金锋等[14] 提出将序列 $\boldsymbol{\omega}$ 块中元素插至所得新序列下界低于 $\boldsymbol{\omega}$ 的最大完工周期 $C_{\max}(\boldsymbol{\omega})$ 的块内. 上述方法的共同缺点是对关键工件考虑不够.

关键工件在 $C_{\max}(\boldsymbol{\omega})$ 的求取中含多个元素，因此经常出现其位置变动对 $C_{\max}(\boldsymbol{\omega})$ 的值变化影响较大的现象(这在本文的仿真计算中得到证实). 在此给出关于关键工件的插入所形成新序列 $\boldsymbol{\omega}_{\text{ins}}$ 的 $C_{\max}(\boldsymbol{\omega}_{\text{ins}})$ 的下界. 记

$$\Delta_{\text{ins}}(p,q)=\max(\delta_b,\delta_a)+a_{m_h u_k}-\delta. \tag{5}$$

其中

$$\delta_b=\sum_{i=m_k+1}^{m_{k+1}}a_{iu_k-1},\ \delta_a=\sum_{i=m_k}^{m_{k+1}-1}a_{iu_k+1},\ \delta=\sum_{i=m_k}^{m_{k+1}}a_{iu_k}.$$

式中：$m_k$ 为工件 $\omega(u_k)$ 所在块 $k$ 所对应的关键机器，$m_h$ 为工件 $\omega(u_k)$ 插入位置 $q$ 所在块 $h$ 对应的关键机器($p=u_k$).

**定理 1**　对序列 $\boldsymbol{\omega}$ 中关键工件 $\omega(u_k)$ 实施插入操作后，得到新序列 $\boldsymbol{\omega}_{\text{ins}}$ 的最大完工期 $C_{\max}(\boldsymbol{\omega}_{\text{ins}})$ 的下界为 $C_{\max}(\boldsymbol{\omega})+\Delta_{\text{ins}}$.

**证明**　式(2) 可改写为关键工件元素与块内工件元素之和，即

$$C_{\max}(\boldsymbol{\omega})=\sum_{k=0}^{l}\sum_{i=m_k}^{m_{k+1}}a_{iu_k}+\sum_{k=1}^{l}\sum_{j=u_{k-1}+1}^{u_k-1}a_{m_k j}. \tag{6}$$

式(6) 恒有 $m_0=1, m_{l+1}=m$. 若 Flow-shop 的第一台机器为关键机器，则有 $m_0=m_1=1$. 若其最后一台机器为关键机器，则有 $m_l=m_{l+1}=m$. 设所插入位置 $q$ 在原序列中属于第 $h$ 块，则有

$$\begin{aligned}&C_{\max}(\boldsymbol{\omega})+\Delta_{\text{ins}}=\\&\sum_{i=m_0}^{m_1}a_{iu_0}+\cdots+\sum_{i=m_l}^{m_{l+1}}a_{iu_l}+\sum_{j=u_0+1}^{u_1-1}a_{m_1 j}+\cdots+\\&\sum_{j=u_{l-1}+\cdots+1}^{u_h-1}a_{m_l j}+\cdots+\max(\delta_b,\delta_a)+a_{m_h u_k}-\sum_{i=m_k}^{m_{k+1}}a_{iu_k}.\end{aligned} \tag{7}$$

当 $\delta_a\geqslant\delta_b$ 时，式(7) 为

$$\begin{aligned}&C_{\max}(\boldsymbol{\omega})+\Delta_{\text{ins}}=\\&\sum_{i=m_0}^{m_1}a_{iu_0}+\cdots+\sum_{i=m_l}^{m_{l+1}}a_{iu_l}+\sum_{j=u_0+1}^{u_1-1}a_{m_1 j}+\\&\cdots+\sum_{j=u_k+2}^{u_{k+1}-1}a_{m_{k+1}j}+\cdots+\sum_{j=u_{h-1}+1}^{u_h-1}a_{m_h j}+\\&\cdots+\sum_{j=u_{l-1}+1}^{u_l-1}a_{m_l j}.\end{aligned} \tag{8}$$

在新序列 $\boldsymbol{\omega}_{\text{ins}}$ 的工时矩阵 $A(\boldsymbol{\omega}_{\text{ins}})$ 中，将属于式(8) 的元素用线段按次序连接起来，可形成一条可行线，该可行线与原序列 $\boldsymbol{\omega}$ 的最大可行线的区别在于：$\omega(u_k)$ 不再是关键工件，而被 $\omega(u_k+1)$ 取代，同时第 $k+1$ 块减少一块内元素 $\omega(u_k+1)$，第 $h$ 块增加一块内元素 $\omega(u_k)$. 显然，该可行线的可行和小于等于 $\boldsymbol{\omega}_{\text{ins}}$ 的最大可行和 $C_{\max}(\boldsymbol{\omega}_{\text{ins}})$，即

$$C_{\max}(\boldsymbol{\omega}_{\text{ins}})\geqslant C_{\max}(\boldsymbol{\omega})+\Delta_{\text{ins}}. \tag{9}$$

同理可证当 $\delta_b\geqslant\delta_a$ 时，式(9) 也成立. □

由定理 1 可初步确定关键工件 $\omega(u_k)$ 的插入位置范围，即仅插入 $\Delta_{\text{ins}}<0$ 的块中，这样 $C_{\max}(\boldsymbol{\omega}_{\text{ins}})$ 才可能低于 $C_{\max}(\boldsymbol{\omega})$. 由此关键工件 $\omega(u_k)$ 的插入位置范围可被缩小. 对块内元素插入位置范围可通过相似定理加以确定，具体可参见文献[13,14].

综上可知，若通过插入操作求最大完工期 $C_{\max}(\boldsymbol{\omega}_{\text{ins}})$ 低于 $C_{\max}(\boldsymbol{\omega})$ 的新序列 $\boldsymbol{\omega}_{\text{ins}}$，则搜索范围可限于一较小的集合，不必对整个邻域 $N_{\text{ins}}$ 中 $(n-1)^2$ 个元素进行搜索.

### 2.2.2　交换操作

记 $s=\langle p,q\rangle(p,q\in(1,2,\cdots,n),p\neq q)$ 为排列 $\boldsymbol{\omega}$ 的交换操作，即将 $\boldsymbol{\omega}$ 中处于第 $p$ 个位置工件 $\omega(p)$ 和第 $q$ 个位置工件 $\omega(q)$ 交换位置，得到一个新的解序列 $\boldsymbol{\omega}_s$，可表示为

$$\boldsymbol{\omega}_s=(\omega(1),\cdots,\omega(q),\cdots,\omega(p),\cdots,\omega(n)).$$

显然，交换操作符合交换律(即 $\langle p,q\rangle=\langle q,p\rangle$)，而插入操作则不符合. 由交换操作所形成 $\boldsymbol{\omega}_s$ 的邻域记为 $N_s$，有

$$N_s(\boldsymbol{\omega})=\{\boldsymbol{\omega}_s\mid s\in S\}. \tag{10}$$

其中

$$S=\{\langle p,q\rangle\mid\mid p-q\mid>1,p,q\in\{1,2,\cdots,n\}\}.$$

式中 $p-q\neq1$，以免与邻域 $N_{\text{ins}}$ 中元素重复.

邻域 $N_s$ 所含元素数目为 $(n-1)(n-2)/2$，虽仅接近邻域 $N_{\text{ins}}$ 元素数目的一半，但对整个邻域进行评估的时间复杂度仍为 $O(mn^3)$. 为缩小搜索范围，可采用前述思路. 即对该操作所产生新序列 $\boldsymbol{\omega}_s$ 的最大完工期 $C_{\max}(\boldsymbol{\omega}_s)$ 的下界进行评估，如可行则继续对 $C_{\max}(\boldsymbol{\omega}_s)$ 进行评估.

需要说明的是，由于交换操作的应用没有插入操作广泛，关于其下界的讨论尚未见于文献. 在此给出对其下界的讨论结果. 记

$$\Delta_s(p,q)=\delta_1-\delta_2. \tag{11}$$

其中

$$\delta_1=\begin{cases}a_{m_k q}+a_{m_h p},\omega(p)\in B_{k\text{ in}},\ \omega(q)\in B_{h\text{ in}};\\\sum_{i=m_k}^{m_{k+1}}a_{iq}+a_{m_h p},p=u_k,\omega(q)\in B_{h\text{ in}};\\\sum_{i=m_k}^{m_{k+1}}a_{iq}+\sum_{i=m_h}^{m_{h+1}}a_{ip},\ p=u_k,\ q=u_h.\end{cases}$$

$$\delta_2=\begin{cases}a_{m_kp}+a_{m_hq},\omega(p)\in B_{k\text{ in}},\omega(q)\in B_{h\text{ in}};\\ \sum\limits_{i=m_k}^{m_{k+1}}a_{ip}+a_{m_hq},p=u_k,\omega(q)\in B_{h\text{ in}};\\ \sum\limits_{i=m_k}^{m_{k+1}}a_{ip}+\sum\limits_{i=m_h}^{m_{h+1}}a_{iq},\ p=u_k,\ q=u_h.\end{cases}$$

**定理 2** 对序列 $\boldsymbol{\omega}$ 中工件 $\omega(p)$ 和 $\omega(q)$ 实施交换操作后，得到新序列 $\boldsymbol{\omega}_s$ 的最大完工期 $C_{\max}(\boldsymbol{\omega}_s)$ 的下界为 $C_{\max}(\boldsymbol{\omega})+\Delta_s$.

**证明** 按式(11)中 $\delta_1,\delta_2$ 和 $\Delta_s$ 的定义，可分 3 种情况进行讨论：1) 交换两工件均为块内工件；2) 交换两工件之一为块内工件，另一为关键工件；3) 交换两工件均为关键工件. 在此仅对情况 3) 进行证明，其余两种情况的证明与此类似.

$$\begin{aligned}&C_{\max}(\boldsymbol{\omega})+\Delta_s=\\&\sum_{i=m_0}^{m_1}a_{iu_0}+\cdots+\sum_{i=m_k}^{m_{k+1}}a_{iu_k}+\cdots+\sum_{i=m_h}^{m_{h+1}}a_{iu_h}+\\&\cdots+\sum_{i=m_l}^{m_{l+1}}a_{iu_l}+\sum_{k=1}^{l}\sum_{j=u_{k-1}+1}^{u_k-1}a_{m_kj}+\sum_{i=m_h}^{m_{h+1}}a_{ip}+\\&\sum_{i=m_k}^{m_{k+1}}a_{iq}-\sum_{i=m_k}^{m_{k+1}}a_{ip}-\sum_{i=m_h}^{m_{h+1}}a_{iq}.\end{aligned}\tag{12}$$

其中有关 $m_0,m_1,m_l$ 和 $m_{l+1}$ 的说明同式(6). 若 $p=u_k,q=u_h$，则有

$$\begin{aligned}&C_{\max}(\boldsymbol{\omega})+\Delta_s=\\&\sum_{i=m_0}^{m_1}a_{iu_0}+\cdots+\sum_{i=m_k}^{m_{k+1}}a_{iq}+\cdots+\sum_{i=m_h}^{m_{h+1}}a_{ip}+\\&\cdots+\sum_{i=m_l}^{m_{l+1}}a_{iu_l}+\sum_{k=1}^{l}\sum_{j=u_{k-1}+1}^{u_k-1}a_{m_kj}.\end{aligned}\tag{13}$$

在新序列 $\boldsymbol{\omega}_s$ 的工时矩阵 $A(\boldsymbol{\omega}_s)$ 中，将属于式(13)的元素用线段按次序连接起来，可形成一条可行线，虽然该可行线与原序列 $\boldsymbol{\omega}$ 的最大可行线形状相同，但所含元素已改变. 显然，该可行线的可行和小于等于 $\boldsymbol{\omega}_s$ 的最大可行和 $C_{\max}(\boldsymbol{\omega}_s)$，即

$$C_{\max}(\boldsymbol{\omega}_s)\geqslant C_{\max}(\boldsymbol{\omega})+\Delta_s.\tag{14}$$

由此定理 2 成立. □

### 2.3 PFSP 求解与指派问题

指派问题是一类特殊的 0-1 规划问题，它要求将 $n$ 个元素恰当地分配到 $n$ 个位置上，使目标函数取得最优值. Kuhn 提出的匈牙利算法，被认为是求解该问题的有效方法，但其前提是必须存在指派矩阵(表示各元素在各个位置上权重(如效率、时间等因素)的矩阵).

要将 PFSP 的求解转化为指派问题来求解，困难在于缺乏一个能准确反映各工件在各个位置上权重的指派矩阵. 为此，本文构造一个矩阵 $W$，该矩阵在算法中不断更新，可近似反映各工件在不同位置上的权重. 具体方法如下：

1) 确定一初始序列 $\boldsymbol{\omega}_0$，$C_{\max}(\boldsymbol{\omega}_0)=c_0$，相应权重矩阵 $W^0$ 中各元素为

$$w_{ij}^0=\begin{cases}1,\ \omega(j)=i;\\0,\ \omega(j)\neq i.\end{cases}\tag{15}$$

2) 如果存在 $n$ 个序列 $\omega_k(1\leqslant k\leqslant n)$，$C_{\max}(\omega_k)=c_k<c_0$，则相应权重矩阵 $W^k$ 中各元素值为

$$w_{ij}^k=\begin{cases}\dfrac{2c_0-c_k}{c_0},\ \omega(j)=i;\\0,\ \omega(j)\neq i.\end{cases}\tag{16}$$

3) 权重矩阵为

$$W=\sum_{k=0}^{n}W_i.\tag{17}$$

式(17)表示 $W$ 记录了初始序列 $\boldsymbol{\omega}_0$ 和优于 $\boldsymbol{\omega}_0$ 的序列系列 $\boldsymbol{\omega}_k(1\leqslant k\leqslant n)$ 的信息，该特征与遗传算法类似. $W$ 的元素 $w_{ij}$ 是 $n+1$ 个元素之和，其值越大，表明在优于 $\omega_0$ 的序列中，工件 $i$ 越可能位于第 $j$ 个位置. 因此可将 $W$ 作为指派矩阵，以最大值作为优化目标，得到一个解序列 $\boldsymbol{\omega}_a$. 通过数值仿真发现，序列 $\boldsymbol{\omega}_a$ 的性能往往优于序列 $\boldsymbol{\omega}_k$ 的性能.

## 3 变路径深度优先搜索算法

深度优先搜索算法是既带有系统性又带有跳跃性的一种搜索算法. 它从问题解空间树的根节点开始搜索，当搜索到树中某一节点时，则判断该节点是否包含问题的解；如不包含，则继续其兄弟节点或父辈节点；否则，对该节点子代进行搜索.

深度优先搜索算法的基本要素包括：初始解、子代产生机制、搜索策略、终止条件设定等. 其中子代产生机制和搜索策略对算法的寻优性能和搜索效率影响很大. 本文提出一种可变路径的深度优先搜索(VPDS)算法，将搜索范围限制在性能较好的子代，压缩搜索空间，同时记录搜索过的节点特性，以便在必要时跳出局部最优. 下面对算法各个要素分别予以阐述.

### 3.1 初 始 解

初始解即搜索树的根节点，以 NEH 算法构造的解作为初始解. 利用 Taillard[20] 的改进方法，使该算法的时间复杂度降为 $O(mn^2)$.

### 3.2 子代产生机制

在空间搜索树上，节点对应某一序列 $\boldsymbol{\omega}$. 设当前节点为序列 $\boldsymbol{\omega}$，则其子节点可按如下方法产生：

1) 对于工件 $\boldsymbol{\omega}(p)$，$1\leqslant p\leqslant n$，若操作 $(p,q)$ 满足 $\Delta_{\text{ins}}(p,q)<0$，且 $(p,q)\in I$，则评估由此产生的新序列 $\omega_{\text{ins}}$. 若满足 $C_{\max}(\boldsymbol{\omega}_{\text{ins}})<C_{\max}(\boldsymbol{\omega})$，则取 $\min C_{\max}(\boldsymbol{\omega}_{\text{ins}})$ 作为其子节点，这样当前节点最多有

$n$ 个子节点.

2) 若在 1) 中无法产生子节点，则将其搜索邻域由 $N_{\text{ins}}$ 转为 $N_s$. 即对于工件 $\boldsymbol{\omega}(p)$，$1 \leqslant p \leqslant n$，考察操作 $\langle p,q\rangle$，若满足 $\Delta_s(p,q)<0$，$\langle p,q\rangle \in S$，则评估由此产生的新序列 $\boldsymbol{\omega}_s$. 若 $C_{\max}(\boldsymbol{\omega}_s)<C_{\max}(\boldsymbol{\omega})$，则取 $\min C_{\max}(\boldsymbol{\omega}_s)$ 作为其子节点.

### 3.3　搜索策略

记当前节点序列为 $\boldsymbol{\omega}$，其子节点为 $\boldsymbol{\omega}'_k$，$1 \leqslant k \leqslant n$. 可选 $\omega'_i$ 满足 $C_{\max}(\omega'_i)=\min C_{\max}(\omega'_k)$ 作为 $\boldsymbol{\omega}$ 的继承节点，对其子代继续搜索. 若当前节点无子节点，则采取以下两种方法跳出局部最优：

1) 回溯方法. 在 $\boldsymbol{\omega}$ 的兄弟节点中按 3.2 节方法寻找符合条件的子节点. 若在兄弟节点中找不到，则回溯至父辈节点寻找；若仍无符合条件的子节点，则转 2).

2) 在 1) 中搜索不到合适子节点的情况下，可用在搜索过程中得到的权重矩阵 $W$，将其转化为指派问题来求解. 以所得新序列 $\boldsymbol{\omega}_a$ 作为当前节点的子节点.

### 3.4　终止条件

在当前节点 $\boldsymbol{\omega}$ 处将问题转化为指派问题，若所得新序列 $\boldsymbol{\omega}_a$ 按 3.2 节方法仍找不到合适的子节点，则 $\boldsymbol{\omega}$ 即为最优解，算法终止.

### 3.5　算法总体流程

输入：工时矩阵 $A$，最好解 $\boldsymbol{\omega}_b=\text{Null}$，当前解 $\boldsymbol{\omega}_c=\text{Null}$，$C_c=0$，权重矩阵 $W=0$；

输出：最好解 $\boldsymbol{\omega}_b$.

Step1：用 NEH 算法获取初始解 $\boldsymbol{\omega}_0$，置 $\boldsymbol{\omega}_b=\boldsymbol{\omega}_0$，更新 $W$.

Step2：对 $\boldsymbol{\omega}_c$ 进行赋值，即 $\boldsymbol{\omega}_c=\boldsymbol{\omega}_b$，$\boldsymbol{C}_c=C_{\max}(\boldsymbol{\omega}_c)$.

Step3：以 $\boldsymbol{\omega}_c$ 为当前节点，按 3.2 节方法产生其子节点 $\boldsymbol{\omega}'_k$，$1 \leqslant k \leqslant n$. 若 $\boldsymbol{\omega}'_k$ 存在，则更新 $W$. 若 $\boldsymbol{\omega}'_i$ 满足 $C_{\max}(\omega'_i)=\min C_{\max}(\boldsymbol{\omega}'_k)$，则置 $\boldsymbol{\omega}_b=\boldsymbol{\omega}'_i$，转 Step2；否则，转 Step4.

Step4：以 $\boldsymbol{\omega}_c$ 的兄弟节点 $\boldsymbol{\omega}_{cb}$ 为当前节点，置 $C_{\max}(\boldsymbol{\omega}_{cb})=C_{\max}(\boldsymbol{\omega}_c)$. 按 3.2 节方法产生其子节点. 若子节点存在，则更新 $W$，以其中 $C_{\max}$ 最小者为 $\boldsymbol{\omega}_b$，转 Step2；否则，转 Step5.

Step5：以 $\boldsymbol{\omega}_c$ 的父辈兄弟节点 $\boldsymbol{\omega}_{cfb}$ 为当前节点，置 $C_{\max}(\boldsymbol{\omega}_{cfb})=C_{\max}(\boldsymbol{\omega}_c)$. 按 3.2 节方法产生其子节点. 若子节点存在，则更新 $W$，以其中 $C_{\max}$ 最小者为 $\boldsymbol{\omega}_b$，转 Step2；否则，转 Step6.

Step6：以 $W$ 作为指派矩阵，将其转化为指派问题求解. 若所求得解 $\boldsymbol{\omega}_a$ 满足 $C_{\max}(\boldsymbol{\omega}_a)<C_{\max}(\boldsymbol{\omega}_c)$，则置 $\boldsymbol{\omega}_b=\boldsymbol{\omega}_a$，转 Step2；否则，以 $\boldsymbol{\omega}_a$ 作为当前节点，置 $C_{\max}(\boldsymbol{\omega}_a)=C_{\max}(\boldsymbol{\omega}_c)$，按 3.2 节方法产生其子节点. 若子节点存在，则更新 $W$，以其中 $C_{\max}$ 最小者为 $\boldsymbol{\omega}_b$，转 Step2；否则，转 Step7.

Step7：算法终止，输出 $\boldsymbol{\omega}_b$.

### 3.6　算法复杂度分析

在 3.5 节的各个步骤中，Step1 中 NEH 算法的时间复杂度为 $O(mn^2)$；Step3 中在最坏情况下（即对整个邻域进行评估）的时间复杂度为 $O(mn^3)$；同样，Step4 ～ Step6 的时间复杂度也为 $O(mn^3)$. 整个算法复杂度既与每个节点中块的性质有关，又与搜索深度和跳出局部最优的情况有关，难以作出精确评估. 如果搜索深度小于某一常数，则算法复杂度不超过 $O(mn^3)$.

## 4　数值仿真结果

本文算法用 Matlab7.0 编程实现，并在 CPU 为 Celeron M(1.6GHz)，内存为 504M 的 PC 机进行测试. 测试算例为 Taillard[21] 提供的典型算例 (Benchmark problems)，共 120 个，按其规模分为 12 类，分别为 $(n\times m)$：$20\times5$，$20\times10$，$20\times20$，$50\times5$，$50\times10$，$50\times20$，$100\times5$，$100\times10$，$100\times20$，$200\times10$，$200\times20$，$500\times20$. 每类包含 10 个具体问题.

从精度和效率综合考虑，PFSP 目前较好的近似算法是 TSGW 方法[13]；单从精度考虑，较好的方法是 RY 方法[16]. 在此从精度和效率两方面考虑，将本文方法 (VPDS) 与 TSGW 方法和 RY 方法的仿真结果进行比较分析. 精度方面衡量指标为相对偏差

$$\text{PRD}(H)=\frac{C_{\max}^{H}-C_{\max}^{T}}{C_{\max}^{T}}. \tag{18}$$

其中：$C_{\max}^{H}$ 为算法 $H$ 的最好解，$H \in \{\text{TSGW},\text{RY},\text{VPDS}\}$；$C_{\max}^{T}$ 为 Taillard[21] 给出的最好解.

对于 12 类问题，3 种算法的 PRD($H$) 如图 1 所示，其中横坐标表示不同规模的算例，顺序同前. 从图中可以看出，VPDS 算法对于大部分算例的求解精度优于 TSGW 算法，接近于 RY 算法，在个别类型算例上所得结果优于 RY 算法.

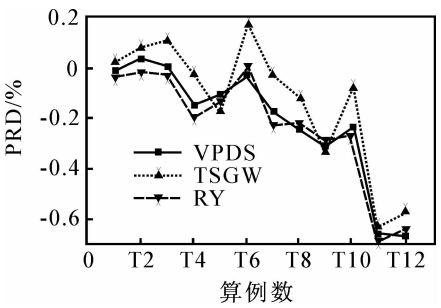


**图 1　各算法对应的 PRD 值**

采用问题求解所需 CPU 时间来衡量算法效率. 规模较大的后 6 类问题求解所需 CPU 时间如图 2 所

示.可以看出,随着问题规模的增长,RY 算法所需时间迅速增长,而 VPDS 算法和 TSGW 算法的增长速度总体上相对较慢.VPDS 算法所需时间略低于 TSGW 算法,问题规模较大时差别较明显.

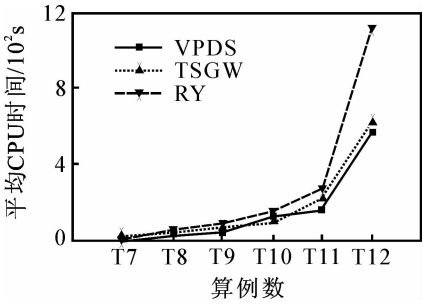


**图 2 各算法求解所需平均 CPU 时间**

在仿真过程中发现,随着问题规模的增大,求解过程中将其转化为指派问题跳出局部最优的次数也迅速增加.为此,分别对算法作了含转化为指派问题环节(记为 VPDS)和不含该环节(记为 VPDS′)的仿真计算.二者的精度比较结果如图 3 所示.对于 12 类典型算例,以 PRD 为精度衡量指标,VPDS 解的精度始终高于 VPDS′ 解,且随着问题规模的增大,算法 VPDS 与 VPDS′ 间解的精度差距不断加大.

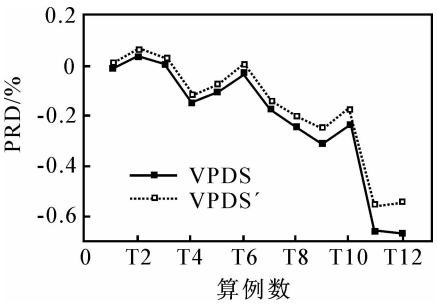


**图 3 算法 VPDS 与 VPDS′ 的 PRD 值比较**

综上可知,从综合性能方面衡量,VPDS 算法明显优于其他算法.特别是对于大规模问题求解(如 500 × 20),其求解精度稍好于 RY 算法,明显好于 TSGW 算法;求解时间远小于 RY 算法(仅为 RY 算法的 50%,即 580 s),略低于 TSGW 算法.通过引入含指派问题环节,进一步提高了算法精度.

## 5 结 论

本文提出一种基于 PFSP 性质的变路径深度优先搜索算法.算法从缩小邻域的角度出发,在充分利用所拓展 PFSP 性质的基础上,获取邻域中工件最大完工期的下界,将搜索范围限制在邻域中最可能的区域,有效地减少了算法计算量.在必要时,可将 PFSP 转化为一个指派问题,自动变更搜索路径,避免陷入局部最优,从而提高算法的精度.数值仿真结果表明,用该算法对大规模 PFSP 问题进行求解,可在较短时间内获得良好的近优解.

本文提出将 PFSP 转化为指派问题的方法,也为其他排序问题(如 TSP 问题)提供了一种新思路.对此可作进一步探讨.

## 参考文献(References)

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

[2] Haouari M, Ladhari T. A branch and bound based local search for the flow-shop problem[J]. J of Operational Research Society, 2003, 54(10): 1076-1084.

[3] 越民义, 韩继业. 同顺序 $m\times n$ 排序问题的一种新方法[J]. 科学通报, 1979, 24(18): 821-824.
(Yue M Y, Han J Y. A new method for the $m\times n$ flow-shop sequencing problem[J]. Chinese Science Bulletin, 1979, 24(18): 821-824.)

[4] 越民义, 韩继业. $n$ 个零件在 $m$ 台机床上的加工顺序问题[J]. 中国科学, 1975, (5): 462-470.
(Yue M Y, Han J Y. On the $n$ job, $m$ machine sequencing problem of flow-shop[J]. Science in China, 1975, (5): 462-470.)

[5] Carlier J, Rebai I. Two branch and bound algorithms for the permutation flow-shop problem[J]. European J of Operational Research, 1996, 90(1): 238-251.

[6] Campbell H G, Dudek R A, Smith M L. A heuristic algorithm for the $n$ job, $m$ machine sequencing problem [J]. Management Science, 1970, 16(10): 630-637.

[7] Nawaz M, Enscore E E, Ham I. A heuristic algorithm for the $m$ machine, $n$ job sequencing problem [J]. Omega, 1983, 11(1): 91-95.

[8] Suliman S. A two-phase heuristic approach to the permutation flow-shop scheduling problem [J]. J of Production Economics, 2000, 64(1): 143-152.

[9] Woo D S, Yim H S. A heuristic algorithm for mean flowtime objective in flow-shop scheduling [J]. Computers and Operations Research, 1998, 25(3): 175-182.

[10] Osman I H, Potts C N. Simulated annealing for permutation flow-shop scheduling[J]. Omega, 1989, 17(6): 551-557.

[11] Ogbu F, Smith D. Simulated annealing for permutation flowshop problem[J]. Omega, 1990, 19(1): 64-67.

[12] Nowicki E, Smutnicki C. A fast tabu search algorithm for the permutation flow-shop problem[J]. European J of Operational Research, 1996, 91(1): 160-175.

(下转第 1213 页)