计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
47(4): 653-662, 2010

有准备时间无等待流水车间调度的搜索算法

王初阳[1,2] 李小平[1,2] 王 茜[1,2] 苑迎春[3]
[1](东南大学计算机科学与工程学院 南京 210096)
[2](计算机网络和信息集成教育部重点实验室(东南大学) 南京 210096)
[3](河北农业大学信息科学与技术学院 河北保定 071001)
(wangcyonline2000@yahoo.com.cn)

A New Local Search Algorithm for No-Wait Fowshops with Setup Time

Wang Chuyang[1,2], Li Xiaoping[1,2], Wang Qian[1,2], and Yuan Yingchun[3]
[1](*School of Computer Science and Engineering, Southeast University, Nanjing* 210096)
[2](*Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing* 210096)
[3](*Faculty of Information Science and Technology, Agriculture University of Hebei, Baoding, Hebei* 071001)

Abstract A new local search algorithm called IVNS (iterated variable neighborhood search) is proposed for the no-wait flowshop scheduling problem with setup time to minimize the total completion time. Three key factors are taken into consideration when designing local search algorithms like IVNS: neighborhoods, neighboring solution evaluations and strategies for escaping local optima. Firstly, three new neighborhoods with larger sizes are introduced to enhance the chance of finding high quality solutions. The neighborhoods are based on job block exchange and have sizes of $O(n^3)$ or $O(n^4)$, larger than the commonly-used insertion and exchange neighborhoods. Secondly, an objective increment method is adopted to speed up the evaluation of neighboring solutions in the neighborhoods, leading to two neighboring solutions compared in constant time and the neighborhoods completely explored in times proportional to their sizes. The objective increment method also reduces the time complexity of the famous NEH algorithm by one order. Finally, IVNS tries to escape local optima by switching between different neighborhoods as well as restarting from perturbation of local optima. IVNS is compared with the best known algorithms for the considered problem on 5400 benchmark instances under identical CPU times. Statistic analysis of the experiment results verifies that IVNS remarkably outperforms the reference algorithms in average solution quality.

Key words local search; no-wait flowshop; scheduling; setup time; total completion time; neighborhood

摘 要 利用迭代变化邻域搜索算法(IVNS)求解最小化总完工时间的有准备时间无等待流水车间调度问题. 设计局部搜索算法需要考虑3个关键因素:所用邻域、解评估和局部最优的克服. 因此,定义了3个较大规模邻域以扩大搜索范围. 为加速解评估,利用目标增量来避免重新计算每个解的目标函数值,使相邻解比较只需常量时间,NEH插入算法的时间复杂度降低一阶. IVNS通过切换邻域和扰动重启,来克服局部搜索易于陷入局部最优解的缺点. 通过与求解该问题的当前最好算法在5400个标准算上,以相同CPU时间进行的实算比较,实验结果统计分析验证了IVNS的寻优性能明显优于参照算法.

关键词 局部搜索;无等待流水车间;调度;准备时间;总完工时间;邻域

中图法分类号 TP278

收稿日期:2008-11-27;**修回日期**:2009-08-17

基金项目:国家"八六三"高技术研究发展计划基金项目(2008AA04Z103);国家自然科学基金项目(60504029,60672092,60873236);河北省自然科学基金项目(F2009000653)

0 引 言

传统流水车间调度假设机器间有无限大缓冲区并且工件加工时间包含工件在机器上的准备时间.这两个假设限制了流水车间在某些工业的应用.如食品工业中,生产技术要求食品烹饪之后马上包装以保持食品新鲜,中间不允许等待.这样的约束称为无等待(no-wait)约束[1].机器间无缓冲区也可造成无等待约束,如机器人单元(robotic cell)组成的生产系统.因此无等待流水车间广泛存在于冶金、塑料、化工等传统工业以及JIT、柔性制造等现代高级制造环境中.当工件在机器上的准备时间很短,可以忽略不计;否则将工件的准备时间从加工时间独立出来进行优化,可缩短加工周期[2].

本文考虑有准备时间的无等待流水车间调度问题,优化目标为最小化总完工时间[3].该问题可简单描述如下.给定 n 个工件和 m 台机器,所有工件在 m 台机器上的加工次序相同(流水车间).已知工件在机器上的加工和准备时间.假设每台机器任意时刻最多能加工1个工件,1个工件任意时刻只能在1台机器上加工;工件的加工是连续的,即在1台机器上加工完成之后立即开始在下一台机器上加工,中间不允许出现等待.目的是确定工件的一个加工顺序使总完工时间最小.

最小化总完工时间的有准备时间无等待流水车间调度问题用 $\alpha/\beta/\gamma$ 三域记法[4]可表示为 $Fm/nwt, S_{ij}/\sum C_i$,其中 α 域表示机器的数量、类型和环境,这里 F 代表流水车间,m 表示机器个数;β 域表示约束条件,这里 nwt 表示无等待约束,S_{ij} 表示工件的准备时间;γ 域表示优化目标,这里是 $\sum C_i$ 即总完工时间,其中 C_i 表示工件 i 的完工时间.当 $m\geqslant 2$ 时,$Fm/nwt, S_{ij}/\sum C_i$ 是NP难问题[5].目前文献多针对2机或3机情况进行优化.Aldowaisan和Allahverdi[5]首先针对 $F2/nwt, S_{ij}/\sum C_i$ 问题给出一个支配规则,并提出了一个分支定界算法和一个构造式启发算法.给定一个工件排列,支配规则对相邻两工件的加工和准备时间进行分析,如果它们满足某些条件,则可判定交换这两个工件的加工次序一定能减少总完工时间.后来他们改进了支配规则[6-7]并推广到3机情况[8].Shyu等人[9]将 $F2/nwt, S_{ij}/\sum C_i$ 转换为时间依赖的旅行售货商问题[10],并用蚁群优化算法进行求解.Brown等人[11]则将 $Fm/nwt, S_{ij}/\sum C_i$ 问题转换为时间依赖的旅行售货商问题,并设计了一个构造启发式算法.最近Ruiz和Allahverdi[3]将支配规则推广到4机情况,并提出5个启发式算法,两个局部搜索算法求解 $Fm/nwt, S_{ij}/\sum C_i$ 问题.Ruiz和Allahverdi[3]提出的局部搜索算法H6和ILS_RA是求解 $Fm/nwt, S_{ij}/\sum C_i$ 问题当前最好的元启发算法.

针对 $Fm/nwt, S_{ij}/\sum C_i$ 问题,本文提出了一个局部搜索算法(iterated variable neighborhood search, IVNS).在应用局部搜索时通常要考虑3个关键因素,即邻域设计、解评估和局部最优的克服.首先,邻域规模越大则越有可能搜索到高质量的解.本文定义了3个较大规模邻域.它们基于工件块交换生成相邻解,规模为 $O(n^3)$ 或 $O(n^4)$,比常用的交换邻域和插入邻域的规模($O(n^2)$)大很多,因而能在更大范围内搜索高质量解.其次由于局部搜索算法是基于相邻解的比较,若先计算两个相邻解的目标函数值后比较相邻解,则需 $O(n)$ 时间,这将成为局部搜索算法的时间瓶颈,尤其当邻域规模较大时.Ruiz和Allahverdi[3]在实验中观察到这个问题,并用支配规则来减少解评估次数.但支配规则只能针对机器数较少的情况进行分析,不具有通用性.本文采用Li等人[12-13]提出的目标增量方法加速解评估.目标增量方法通过分析邻域基本操作对目标函数值的影响,来计算一个解 π 的相邻解 π' 相对于 π 的目标增量.若目标增量小于零,则 π' 优于 π.因目标增量的计算只需 $O(1)$ 时间,所以可以加速解评估和邻域搜索.由于邻域基本操作的目标增量与机器个数无关,因此可以使用于任何机器数的情况,比基于支配规则的加速方法更具通用性.此外目标增量还可用来加速NEH[14]插入算法.最后IVNS通过切换邻域[15]和扰动重启[16]两种方法克服局部搜索算法易于陷入局部最优的缺点.

1 问题描述

给定 n 个工件的集合 $J=\{1,2,\cdots,n\}$ 和 m 台机器的集合 $M=\{1,2,\cdots,m\}$.已知工件 $i(1\leqslant i\leqslant n)$ 在机器 $k(1\leqslant k\leqslant m)$ 上的加工时间 $t_{i,k}$ 和准备时间 $s_{i,k}$.由于无等待约束,一个调度对应工件 $1,2,\cdots,n$ 的一个排列.

为便于计算,引入虚工件0,其加工时间和准备

时间均为0,即 $t_{0,k}=s_{0,k}=0(1\leqslant k\leqslant m)$.一个调度表示为一个可行解(序列) $\pi=(\pi_{[0]},\pi_{[1]},\cdots,\pi_{[n]},\pi_{[n+1]})$,其中,$\pi_{[i]}$ 表示序列中位置 i 上的工件;$\pi_{[0]}$ 和 $\pi_{[n+1]}$ 固定为虚工件0,表示调度的开始和结束;$(\pi_{[1]},\cdots,\pi_{[n]})$是工件 $1,2,\cdots,n$ 的一个排列.

由于无等待约束,只要工件 j 紧邻在工件 i 之后加工,则工件 j 和工件 i 的完工时间之差(后面称之为工件 i 到工件 j 的距离) $d_{i,j}$ 就可由式(1)计算[11]:

$$d_{i,j}=\begin{cases} d_{i,j}^{m},\ 0\leqslant \mathrm{i}\leqslant \mathrm{n},\ 1\leqslant \mathrm{j}\leqslant \mathrm{n},\ \text{且 } \mathrm{i}\neq \mathrm{j};\\ 0,\ 0\leqslant \mathrm{i}\leqslant \mathrm{n},\ \mathrm{j}=0. \end{cases} \tag{1}$$

其中,$d_{i,j}^{m}=\max\{d_{i,j}^{k-1}-t_{i,k},s_{j,k}\}+t_{j,k}$,$1\leqslant k\leqslant m$ 且 $\forall\,\mathrm{i},\mathrm{j},\mathrm{d}_{i,j}^{0}=0$.因此可预先计算任意一对工件间的距离,组成 $(n+1)$ 阶距离方阵 $\boldsymbol{D}=(d_{ij})$.由距离矩阵 $\boldsymbol{D}$,任一个可行解 π 的总完工时间 $TCT(\pi)$ 由式(2)给出:

$$TCT(\pi)=\sum_{i=1}^{n}(n+1-i)\times d_{[i-1,i]}=\sum_{i=1}^{n+1}(n+1-i)\times d_{[i-1,i]}. \tag{2}$$

目的是在所有可行解组成的集合 Π 中寻找一个可行解 π^{*},使

$$TCT(\pi^{*})\leqslant \mathrm{TCT}(\pi),\ \pi\in\Pi. \tag{3}$$

2 新邻域及目标增量分析

邻域基本操作对一个解 π 施加变换产生一些新解,称之为 π 的相邻解,这些新解组成的集合称为 π 的邻域.这里给出3个基于工件块交换的邻域及其基本操作对应的目标增量.一个工件块是一个可行解中两个位置间的工件组成的子序列.下面首先给出邻域基本操作的定义.

定义1. 相邻块交换(adjacent block exchange,ABE):对于给定解 π 中3个位置 i,j,k 且 $0\leqslant i\leqslant n-2,i+1\leqslant j\leqslant n-1,j+1\leqslant k\leqslant n$,交换相邻工件块 $[\pi_{[i+1]},\cdots,\pi_{[j]}]$ 和 $[\pi_{[j+1]},\cdots,\pi_{[k]}]$ 生成相邻解 $\pi'=(\pi_{[0]},\cdots,\pi_{[i]},\pi_{[j+1]},\cdots,\pi_{[k]},\pi_{[i+1]},\cdots,\pi_{[j]},\pi_{[k+1]},\cdots,\pi_{[n+1]})$.

相邻块交换操作记为 $ABE(\pi,i,j,k)$,所有这些相邻解组成的集合称为解 π 的相邻块交换邻域,记作 $N_{\mathrm{ABE}}(\pi)$.

定义2. 不相邻块交换(nonadjacent block exchange,NBE):对于给定解 π 中4个位置 h,i,j,k 且 $0\leqslant h\leqslant n-4,h+1\leqslant i\leqslant n-3,i+2\leqslant j\leqslant n-1,j+1\leqslant k\leqslant n$,交换不相邻工件块 $[\pi_{[h+1]},\cdots,\pi_{[i]}]$ 和 $[\pi_{[j+1]},\cdots,\pi_{[k]}]$ 生成相邻解 $\pi'=(\pi_{[0]},\cdots,\pi_{[h]},\pi_{[j+1]},\cdots,\pi_{[k]},\pi_{[i+1]}\cdots,\pi_{[j]},\pi_{[h+1]},\cdots,\pi_{[i]},\pi_{[k+1]},\cdots,\pi_{[n+1]})$.

不相邻块交换操作记为 $NBE(\pi,h,i,j,k)$,所有这些相邻解组成的集合称为解 π 的不相邻块交换邻域,记作 $N_{\mathrm{NBE}}(\pi)$.因邻域 $N_{\mathrm{ABE}}(\pi)$ 和 $N_{\mathrm{NBE}}(\pi)$ 的规模分别为 $O(n^{3})$ 和 $O(n^{4})$,远大于常用的插入和交换邻域的规模 $O(n^{2})$,搜索到高质量解的概率更大,但邻域搜索时间也相应增加.利用Li等人[12-13]提出的目标增量方法可加速相邻解的比较.邻域基本操作的目标增量(即它们生成的解 π' 相对于原解 π 的目标增量)基于 π 的距离和向量 $\boldsymbol{S}$,如下定义:

$$S_{[\mathrm{k}]}=\begin{cases} 0,\ k=0,\\ \sum_{i=1}^{k}d_{[\mathrm{i}-1,i]},\ 1\leqslant \mathrm{k}\leqslant \mathrm{n}+1. \end{cases} \tag{4}$$

即向量 $\boldsymbol{S}$ 的第 k 个分量 $S_{[\mathrm{k}]}$ 定义为 π 中前 k 个距离之和.通过 $\boldsymbol{S}$ 可计算 π 中任一工件块 $[\pi_{[i+1]},\cdots,\pi_{[j]}]$ 的距离和,即 $\sum_{k=i+2}^{j}d_{[\mathrm{k}-1,k]}=S_{[\mathrm{j}]}-S_{[\mathrm{i}+1]}$.根据式(2),工件块 $[\pi_{[i+1]},\cdots,\pi_{[j]}]$ 在 π 中左移(或右移)一个位置对目标值的贡献增加(或减少) $S_{[j]}-S_{[i+1]}$,因此有如下定理.

定理1. 给定整数 i,j,k 和可行解 π,且 $0\leqslant i\leqslant n-2,i+1\leqslant j\leqslant n-1,j+1\leqslant k\leqslant n$,则相邻块交换操作的目标增量 $\Delta_{ABE(\pi,i,j,k)}=TCT(\pi')-TCT(\pi)$ 可由下面公式计算:

$$\begin{aligned}\Delta_{ABE(\pi,i,j,k)}=&(j-i)\times(S_{[\mathrm{k}]}-S_{[\mathrm{j}+1]})-(k-j)\times(S_{[\mathrm{j}]}-S_{[\mathrm{i}+1]})+(n-i)\times\\&(d_{[\mathrm{i},\mathrm{j}+1]}-d_{[\mathrm{i},\mathrm{i}+1]})+(n-k)\times(d_{[\mathrm{j},\mathrm{k}+1]}-d_{[\mathrm{k},\mathrm{k}+1]})+\\&(n+j-i-k)\times d_{[\mathrm{k},\mathrm{i}+1]}-(n-j)\times d_{[\mathrm{j},\mathrm{j}+1]}.\end{aligned} \tag{5}$$

证明.在 $ABE(\pi,i,j,k)$ 操作中,工件块 $[\pi_{[i+1]},\cdots,\pi_{[j]}]$ 右移了 $\mathrm{k}-j$ 个位置,目标值减少 $(k-j)\times(S_{[j]}-S_{[i+1]})$ 个单位.工件块 $[\pi_{[j+1]},\cdots,\pi_{[k]}]$ 左移了 $\mathrm{j}-i$ 个位置,目标值增加 $(j-i)\times(S_{[k]}-S_{[j+1]})$ 个单位.由于这两个工件块的交换,3个距离被替代:$d_{[i,j+1]}$ 替代了 $d_{[i,i+1]}$ 且系数均为 $n-i$;$d_{[j,k+1]}$ 替代了 $d_{[k,k+1]}$ 且系数均为 $n-k$;系数为 $n-j$ 的距离 $d_{[j,j+1]}$ 被系数为 $n+j-i-k$ 的完工时间差 $d_{[k,i+1]}$ 代替.由此可得式(5). 证毕.

定理2. 给定整数 h,i,j,k 和可行解 π,且 $0\leqslant h\leqslant n-4,h+1\leqslant i\leqslant n-3,i+2\leqslant j\leqslant n-1,j+1\leqslant$

$k \leqslant n$,则不相邻块交换操作的目标增量 $\Delta_{NBE(\pi,h,i,j,k)} = TCT(\pi') - TCT(\pi)$可由下面公式计算:

$$\begin{aligned}\Delta_{NBE(\pi,h,i,j,k)} &= (j-h)\times(S_{[k]} - S_{[j+1]}) - \\ &(k-i)\times(S_{[i]} - S_{[h+1]}) + (i+j-h-k)\times \\ &(S_{[j]} - S_{[i+1]}) + (n-h)\times \\ &(d_{[h,j+1]} - d_{[h,h+1]}) + (n-k)\times \\ &(d_{[i,k+1]} - d_{[k,k+1]}) + (n+j-h-k)\times \\ &d_{[k,i+1]} - (n-i)\times d_{[i,i+1]} + \\ &(n+i-h-k)\times d_{[j,h+1]} - (n-j)\times d_{[j,j+1]}.\end{aligned} \tag{6}$$

证明. 方法同定理 1 的证明.

由以上两个定理,一个相邻解的目标增量可在 $O(1)$时间内求出,因而搜索一个解的整个 N_{ABE} 和 N_{NBE} 邻域的时间和其规模呈线性关系. 邻域 N_{ABE} 的规模为 $O(n^4)$,可如下简化:将一个工件块的长度固定为 1,将得到的邻域称为简化块交换邻域,其规模为 $O(n^3)$,其基本操作记作 $SBE(\pi,i,j,k)$,定义如下.

定义 3. 简化块交换(simplified bock exchange, SBE). 给定解 π 中位置 i 和一个工件块$[\pi_{[j+1]},\cdots,\pi_{[k]}]$,1. 当 $1\leqslant i\leqslant n-3$, $i+2\leqslant j\leqslant n-1$, $i+2\leqslant j\leqslant n-1$, $SBE(\pi,i,j,k)$ 生成相邻解 $\pi'=(\pi_{[0]},\cdots,\pi_{[i-1]},\pi_{[j+1]},\cdots,\pi_{[k]},\pi_{[i+1]},\cdots,\pi_{[j]},\pi_{[i]},\pi_{[k+1]},\cdots,\pi_{[n+1]})$;2. 当 $0\leqslant j\leqslant n-3$, $j+1\leqslant k\leqslant n-2$, $k+2\leqslant i\leqslant n$, $SBE(\pi,i,j,k)$生成相邻解 $\pi'=(\pi_{[0]},\cdots,\pi_{[j]},\pi_{[i]},\pi_{[k+1]},\cdots,\pi_{[i-1]},\pi_{[j+1]},\cdots,\pi_{[k]},\pi_{[i+1]},\cdots,\pi_{[n+1]})$.

所有这些解组成的集合称为 π 的简化块交换邻域,记作 $N_{SBE}(\pi)$. 由定理 2 容易验证 $SBE(\pi,i,j,k)$的目标增量 $\Delta_{SBE(\pi,i,j,k)} = TCT(\pi') - TCT(\pi)$可如下计算:

推论 1. 当 $1\leqslant i\leqslant n-3$, $i+2\leqslant j\leqslant n-1$, $i+2\leqslant j\leqslant n-1$ 时, $\Delta_{SBE(\pi,i,j,k)} = (j-i+1)\times(S_{[k]} - S_{[j+1]}) - (k-j-1)\times(S_{[j]} - S_{[i+1]}) + (n-i+1)\times(d_{[i-1,j+1]} - d_{[i-1,i]}) + (n-k)\times(d_{[i,k+1]} - d_{[k,k+1]}) + (n-k+1)\times d_{[j,i]} - (n-i)\times d_{[i,i+1]} + (n+j-i-k+1)\times d_{[k,i+1]} - (n-j)\times d_{[j,j+1]}$;

而当 $0\leqslant j\leqslant n-3$, $j+1\leqslant k\leqslant n-2$, $k+2\leqslant i\leqslant n$ 时,

$$\begin{aligned}\Delta_{SBE(\pi,i,j,k)} &= (k-j-1)\times(S_{[i-1]} - S_{[k+1]}) - \\ &(i-k)\times(S_{[k]} - S_{[j+1]}) + (k-j-1)\times(S_{[i-1]} - \\ &S_{[k+1]}) - (i-k)\times(S_{[k]} - S_{[j+1]}) + (n-j)\times(d_{[j,i]} - \\ &d_{[j,j+1]}) + (n-i)\times(d_{[k,i+1]} - d_{[i,i+1]}) + (n-j-1)\times \\ &d_{[i,k+1]} - (n-k)\times d_{[k,k+1]} + (n+k-i-j)\times \\ &d_{[i-1,j+1]} - (n-i+1)\times d_{[i-1,i]}.\end{aligned}$$

新邻域及其基本操作的目标增量的意义在于其通用性. 首先,从新邻域可以导出多种简单邻域,即通过限制工件块的长度来设计简单的邻域. 简化块交换邻域(定义 3)给出了一个具体例子. 常用的插入和交换邻域可以看作本文邻域的特例:将相邻块交换操作中的一个块的长度固定为 1,则得到常用的插入邻域;将不相邻块交换操作中的两个块的长度都固定为 1,则得到常用的交换邻域. 即使较少使用的 k-插入邻域[17]也是相邻块交换邻域的特例. 因此新邻域更具通用性,而其目标增量则为这些新邻域提供了一个通用公式,从而不必分别为这些简单邻域分析目标增量. 其次,由于邻域基本操作的目标增量与机器个数无关,因此可以使用于任何机器数的情况,比基于支配规则的加速方法[3]更具通用性. 由于这些新邻域规模较大,可以为搜索算法提供丰富的搜索路径,使搜索算法易于发现高质量的局部最优解,而目标增量又可加快相邻解的评估.

目标增量还可用来加速 NEH 插入算法[14]. 给定一个初始工件顺序 $\pi=(\pi_{[0]},\pi_{[1]},\cdots,\pi_{[n+1]})$, NEH 插入算法的步骤如下:

算法 1. NEH 插入算法.

输入:初始工件序列 $\pi=(\pi_{[0]},\pi_{[1]},\cdots,\pi_{[n+1]})$;

输出:改进的工件序列 π'.

① $\pi'\leftarrow(0,\pi_{[1]},0)$, $i\leftarrow 2$;

② 将工件 $\pi_{[i]}$插入到 π'的位置 k, $k=1,\cdots,i$, 所得部分序列记作 π'_k, $j=\arg\min\limits_{k=1,\cdots,i}\{TCT(\pi'_k)\}$, $\pi'\leftarrow\pi'_j$;

③ 若 $i<n$, $i\leftarrow i+1$, 转②;

④ 返回 π',算法停止.

NEH 插入算法主要是步骤②. 因用式(2)直接计算一个 π'_k 的目标值需 $O(i)$时间,而步骤②需要计算 i 个部分序列的目标值,所以需 $O(i^2)$时间. 整个算法需要执行 n 次步骤②,所以复杂度为 $O(n^3)$. 设 π'的距离和向量为 S',则部分序列 π'_k 相对于 π'的目标增量为 $S'_{[k-1]}+(i-k+1)\times d_{\pi'_{[k-1]},\pi_{[i]}}+(i-k)\times(d_{\pi_{[i]},\pi'_{[k+1]}} - d_{\pi'_{[k]},\pi'_{[k+1]}})$,这样可在 $O(i)$时间计算出所有 i 个部分序列的目标值,所以 NEH 插入算法的计算复杂度降为 $O(n^2)$.

3 迭代变化邻域搜索算法

迭代变化邻域搜索(iterated variable neighborhood search, IVNS)是一种多重启搜索方法,主要思想是从不同的初始解进行局部搜索. 多重启搜索方法包括迭代局部搜索算法[16]、贪心随机自适应搜索过

程[18-19]、自适应记忆规划[20]. IVNS 包含 3 个主要部分:初始解的生成、局部搜索过程、扰动过程.

本文根据最近邻居原则生成初始序列,再使用 NEH 插入算法对其进行改进,将改进解作为局部搜索的初始解.过程如下:

算法 2. 初始解生成算法 ISG.

输入:算例数据;

输出:初始解 π.

① 用式(1)计算距离矩阵 $\boldsymbol{D}$;

② $\pi_{[0]}\leftarrow 0$, $\pi_{[n+1]}\leftarrow 0$, $\pi'\leftarrow\{1,2,\cdots,n\}$, $i\leftarrow 1$;

③ 令 $j=\arg\min_{\gamma\in\pi}\{d_{\pi_{[i-1]},\gamma}\}$, $\pi_{[i]}\leftarrow j$, $\pi'\leftarrow\pi'-\{j\}$;

④ 若 $i<n$, $i\leftarrow i+1$, 转③;

⑤ 对 π 进行 NEH 插入,得到的解仍记作 π;

⑥ 返回 π,算法停止.

用式(1)计算距离矩阵 $\boldsymbol{D}$ 的时间复杂度为 $O(mn^2)$,而最近邻居生成初始序列和 NEH 插入算法均需要 $O(n^2)$,所以 ISG 的时间复杂度为 $O(mn^2)$.

传统局部搜索的每一次迭代都是在当前解的某个邻域中搜索一个改进解,如果找到改进解,则将改进解作为当前解,继续下一次迭代;否则当前解是关于该邻域的局部最优解,搜索结束.本文采用以下过程实现局部搜索的一次迭代,即在当前解的邻域中搜索一个改进解.设所用邻域为 NS_{ABE},当前解为 π,并将该过程记作 $FN_{\mathrm{ABE}}(\pi,S)$,则搜索方法如下:

算法 3. $FN_{\mathrm{ABE}}(\pi,S)$.

输入:当前解 π;

输出:改进了的 π.

① $I\leftarrow\{0,\cdots,n-2\}$;

② 随机取 $i\in I$, $I\leftarrow I-\{i\}$, $J\leftarrow\{i+1,\cdots,n-1\}$;

③ 随机取 $j\in J$, $J\leftarrow J-\{j\}$, $K\leftarrow\{j+1,\cdots,n\}$;

④ 随机取 $k\in K$, $K\leftarrow K-\{j\}$,计算 $\Delta_{ABE(\pi,i,j,k)}$;

⑤ 若 $\Delta_{ABE(\pi,i,j,k)}<0$, $TCT(\pi)\leftarrow TCT(\pi)+\Delta_{ABE(\pi,i,j,k)}$, $\pi\leftarrow ABE(\pi,i,j,k)$,返回 π,算法停止;

⑥ 若 $K\neq\varnothing$,转④;

⑦ 若 $J\neq\varnothing$,转③;

⑧ 若 $I\neq\varnothing$,转②;

⑨ 返回 π,算法停止.

算法 3 在当前解 π 的 N_{ABE} 邻域中以随机顺序搜索,只要发现一个改进解就返回;否则 π 是关于 N_{ABE} 的局部最优解,返回 π.显然算法 3 的最坏时间复杂度为 $O(n^3)$.在 N_{SBE} 中搜索一个改进解的方法类似,并将该过程记作 $FN_{\mathrm{SBE}}(\pi,S)$,其复杂度亦为 $O(n^3)$.

因一个邻域的局部最优解在另一邻域中未必是局部最优的,所以当搜索止于某个邻域的局部最优解时,通过切换邻域可能会使搜索得以继续.基于这种思想,Hansen 和 Mladenovic 提出了变化邻域搜索(variable neighborhood search, VNS)[15].本文采用变化邻域搜索作为 IVNS 的局部搜索过程,其步骤如下:

算法 4. 变化邻域搜索 $VNS(\pi)$;

输入:搜索起点 π;

输出:改进了的 π.

① 计算 π 的距离和向量 $\boldsymbol{S}$;

② $\pi'\leftarrow FN_{\mathrm{ABE}}(\pi,\boldsymbol{S})$;

③ 若 $TCT(\pi')<TCT(\pi)$, $\pi\leftarrow\pi'$,转①;

④ $\pi'\leftarrow FN_{\mathrm{SBE}}(\pi,\boldsymbol{S})$;

⑤ 若 $TCT(\pi')<TCT(\pi)$, $\pi\leftarrow\pi'$,转①;

⑥ 返回 π,算法停止.

以上过程首先在当前解 π 的 N_{ABE} 邻域中以随机顺序寻找一个改进解,如果找到则更新当前解,并继续在 N_{ABE} 邻域中搜索;否则当前解是关于 N_{ABE} 邻域的一个局部最优解,则将邻域切换为 N_{SBE},并在 N_{SBE} 中以随机的顺序搜索改进解.如果在 N_{SBE} 中找到一个改进解,则更新当前解,并将邻域切换回 N_{ABE};如果找不到改进解则算法停止.因此 VNS 的输出是关于解集合 $N_{\mathrm{ABE}}\cup N_{\mathrm{SBE}}$ 的局部最优解.

为进一步扩大搜索范围,可以从一个新的初始解开始 VNS 算法.构造新初始解常用方法有随机构造、扰动构造[17]、贪心随机自适应方法[18-19]、路径重建等[21].本文采用扰动构造方法是对上一轮 VNS 的输出解进行扰动,然后从扰动解重启 VNS 算法,这样搜索的历史信息得到部分保留.扰动过程如下:随机选择两个位置,将其中一个位置上的任务插入到另一个位置,如此进行 p 次.通过初始实验设 p 值为 5.

综合上述内容,整个 IVNS 算法的步骤如下:

算法 5. 迭代变化邻域算法 IVNS.

输入:算例数据文件;

输出:近似最优解 π^*.

① 用 ISG 生成初始解 π;

② $\pi\leftarrow VNS(\pi)$, $\pi^*\leftarrow\pi$;

③ 如果 CPU 时间到,则转⑨;

④ 对 π 进行扰动,得到解 π';

⑤ $\pi'\leftarrow VNS(\pi')$;

⑥ 若 $TCT(\pi')\leqslant TCT(\pi)$,则 $\pi\leftarrow\pi'$;

⑦ 若 $TCT(\pi')<TCT(\pi^*)$,则 $\pi^*\leftarrow\pi'$;

⑧ 转③;

⑨ 输出 π^*,算法停止.

IVNS 是一个迭代搜索过程,它重复执行下述过程:对上一轮 VNS 的输出进行扰动,然后以扰动解为起点,再次进行 VNS 搜索.元启发搜索算法的结束条件通常采用固定代数,固定的 CPU 时间或评估解的数目,目的是使元启发算法的运行时间与问题规模呈多项式关系.为便于比较,本文采用与文献[3]相同的结束条件,即固定 CPU 时间.固定的 CPU 时间作为结束条件使所有被比较算法得到相同的运算时间,因此只需比较算法的平均解质量,所以易于公平比较算法,因此被广泛采用[3,20,22].本文算法运行 CPU 时间设为 $n\times m\times 10$ ms,因此 IVNS 的时间复杂度为 $O(mnC)$,其中 C 是一个指定的常数.

4 实验结果与分析

IVNS 与求解 $Fm|nwt, S_{ij}|\sum C_i$ 的两个当前最好算法 H6 和 ILS_RA[3] 进行实算比较.算例集[3]包含 5 400 个算例.工件的加工时间分别取自均匀分布整数区间 U[1,10]和 U[1,100].当加工时间取自 U[1,10],准备时间分别取自 U[0,5],U[0,10]和 U[0,15];当加工时间取自 U[1,100],准备时间取自 U[0,5],U[0,10]和 U[0,15].算例分属两个子集:小规模子集的 3 000 算例中,$n\in\{15,20,25,30\}$,$m\in\{2,3,4,5,6\}$;大规模子集的 2 400 算例中,$n\in\{50,100,150,200\}$,$m\in\{10,20,30,40\}$.算例的参考目标值通过较长时间运行 ILS_RA 得到.算法用 Java 实现,以相同的 CPU 时间($n\times m\times 10$ ms)作为结束条件.实验运行平台为 Intel Pentium Ⅳ 3.0 GHz 处理器,1 GB 内存的 PC 机.每个算法在单个算例上运算 5 次,用平均相对百分比偏差(average relative percentage deviation,ARPD)度量算法的平均解质量.表 1~4 按算例的规模和加工及准备时间组合给出了 3 个算法的 ARPD.

Table 1 ARPDs for "small" Instances with the Processing Time U[1,10]

表 1 加工时间取自 U[1,10]时 3 个算法在小规模算例集上的 ARPD

n	m	Setup Time from U[0,5]			Setup Time from U[0,10]			Setup Time from U[0,15]		
		H6	ILS	IVNS	H6	ILS	IVNS	H6	ILS	IVNS
15	2	0.02	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00
15	3	0.01	0.00	0.00	0.03	0.02	0.00	0.02	0.01	0.00
15	4	0.02	0.00	0.00	0.05	0.03	0.00	0.02	0.00	0.00
15	5	0.04	0.00	0.00	0.03	0.01	0.00	0.02	0.00	0.00
15	6	0.02	0.00	−0.01	0.01	0.00	−0.01	0.01	0.00	0.00
20	2	0.11	0.01	−0.05	0.10	0.01	−0.01	0.03	0.01	0.00
20	3	0.21	0.02	−0.04	0.11	0.04	−0.03	0.05	0.01	−0.01
20	4	0.11	0.01	−0.02	0.10	0.01	0.00	0.12	0.04	−0.02
20	5	0.18	0.05	−0.03	0.14	0.03	−0.01	0.07	0.03	−0.01
20	6	0.12	0.03	−0.02	0.09	0.02	−0.01	0.08	0.00	−0.02
25	2	0.25	0.02	−0.05	0.14	0.03	−0.09	0.12	0.04	−0.03
25	3	0.23	−0.01	−0.15	0.17	0.05	−0.08	0.17	0.03	−0.12
25	4	0.30	0.07	−0.10	0.14	0.01	−0.06	0.16	0.02	−0.09
25	5	0.27	0.05	−0.16	0.24	0.08	−0.08	0.19	0.04	−0.05
25	6	0.33	0.04	−0.07	0.29	0.06	−0.06	0.27	0.08	−0.04
30	2	0.38	0.01	−0.13	0.21	0.03	−0.14	0.13	0.03	−0.08
30	3	0.40	0.06	−0.22	0.30	0.06	−0.17	0.24	0.07	−0.17
30	4	0.45	0.11	−0.28	0.40	0.07	−0.16	0.24	0.07	−0.15
30	5	0.61	0.14	−0.22	0.44	0.14	−0.14	0.31	0.11	−0.20
30	6	0.60	0.12	−0.14	0.35	0.09	−0.13	0.29	0.09	−0.12
Average		0.23	0.04	−0.08	0.17	0.04	−0.06	0.13	0.03	−0.06

Table 2 ARPDs for "small" Instances with the Processing Time U[1,100]

表 2 加工时间取自 U[1,100]时 3 个算法在小规模算例集上的 ARPD

n	m	Setup Time from U[0,50]			Setup Time from U[0,100]			Setup Time from U[0,150]		
		H6	ILS	IVNS2	H6	ILS	IVNS2	H6	ILS	IVNS2
15	2	0.01	0.00	−0.01	0.03	0.01	−0.01	0.01	0.00	0.00
15	3	0.02	0.01	−0.01	0.03	0.01	0.00	0.01	0.00	0.00
15	4	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15	5	0.01	0.01	0.00	0.03	0.01	0.00	0.03	0.00	0.00
15	6	0.01	0.01	0.00	0.03	0.01	0.00	0.03	0.00	0.00
20	2	0.10	0.01	−0.04	0.10	0.00	−0.01	0.04	0.00	−0.01
20	3	0.14	0.00	−0.08	0.10	0.02	−0.03	0.09	0.02	−0.02
20	4	0.14	0.01	−0.03	0.12	0.01	−0.04	0.09	0.02	−0.03
20	5	0.15	0.07	−0.03	0.13	0.03	−0.02	0.12	0.01	−0.03
20	6	0.14	0.02	−0.01	0.12	0.04	−0.01	0.12	0.02	−0.03
25	2	0.30	0.07	−0.12	0.23	0.05	−0.06	0.14	0.04	−0.09
25	3	0.29	0.07	−0.16	0.24	0.03	−0.13	0.17	0.04	−0.07
25	4	0.36	0.05	−0.18	0.25	0.05	−0.07	0.17	0.03	−0.11
25	5	0.31	0.09	−0.08	0.25	0.06	−0.03	0.21	0.08	−0.05
25	6	0.38	0.14	−0.07	0.22	0.06	−0.07	0.26	0.10	−0.04
30	2	0.36	0.08	−0.19	0.20	0.05	−0.16	0.16	0.07	−0.12
30	3	0.61	0.17	−0.25	0.33	0.14	−0.18	0.24	0.08	−0.15
30	4	0.52	0.17	−0.17	0.36	0.07	−0.13	0.27	0.10	−0.15
30	5	0.55	0.24	−0.10	0.41	0.12	−0.14	0.28	0.11	−0.13
30	6	0.49	0.15	−0.17	0.35	0.18	−0.14	0.33	0.12	−0.12
Average		0.25	0.07	−0.08	0.18	0.05	−0.06	0.14	0.04	−0.06

Table 3 ARPDs for "large" Instances with the Processing Time U[1, 10]

表 3 加工时间取自 U[1, 10]时 3 个算法在大规模算例集上的 ARPD

n	m	Setup Time from U[0,5]			Setup Time from U[0,10]			Setup Time from U[0,15]		
		H6	ILS	IVNS2	H6	ILS	IVNS2	H6	ILS	IVNS2
50	10	0.68	0.22	−0.55	0.65	0.21	−0.55	0.61	0.28	−0.42
50	20	0.66	0.16	−0.42	0.63	0.20	−0.35	0.44	0.17	−0.44
50	30	0.58	0.19	−0.45	0.55	0.18	−0.41	0.51	0.14	−0.38
50	40	0.51	0.10	−0.42	0.50	0.17	−0.41	0.45	0.12	−0.37
100	10	0.64	0.39	−1.55	0.53	0.34	−1.40	0.87	0.72	−0.84
100	20	0.75	0.40	−1.35	0.52	0.35	−1.29	0.71	0.52	−0.91
100	30	0.61	0.40	−1.42	0.51	0.27	−1.30	0.45	0.33	−1.08
100	40	0.53	0.25	−1.45	0.45	0.22	−1.33	0.63	0.42	−0.99
150	10	0.64	0.43	−2.15	0.46	0.45	−1.67	0.63	0.59	−1.25
150	20	0.51	0.38	−1.98	0.37	0.29	−1.66	0.47	0.42	−1.39
150	30	0.60	0.49	−1.84	0.39	0.29	−1.76	0.44	0.37	−1.47
150	40	0.53	0.35	−1.90	0.36	0.30	−1.76	0.52	0.44	−1.37
200	10	0.60	0.53	−2.23	0.52	0.53	−1.76	0.34	0.38	−1.53
200	20	0.51	0.48	−2.14	0.37	0.31	−1.87	0.27	0.28	−1.73
200	30	0.46	0.51	−2.13	0.38	0.38	−1.85	0.29	0.24	−1.77
200	40	0.54	0.57	−2.15	0.33	0.32	−1.96	0.34	0.32	−1.75
Average		0.59	0.36	−1.51	0.47	0.30	−1.33	0.50	0.36	−1.11

Table 4 ARPDs for "large" Instances with the Processing Time U[1, 100]

表 4 加工时间取自 U[1, 100]时 3 个算法在大规模算例集上的 ARPD

n	m	Setup Time from U[0,50]			Setup Time from U[0,100]			Setup Time from U[0,150]		
		H6	ILS	IVNS	H6	ILS	IVNS	H6	ILS	IVNS
50	10	0.62	0.16	−0.59	0.62	0.27	−0.49	0.46	0.15	−0.52
50	20	0.55	0.19	−0.47	0.56	0.22	−0.44	0.49	0.19	−0.37
50	30	0.56	0.13	−0.54	0.57	0.18	−0.32	0.46	0.12	−0.41
50	40	0.53	0.07	−0.45	0.47	0.16	−0.42	0.44	0.16	−0.44
100	10	0.61	0.36	−1.51	0.50	0.30	−1.40	0.37	0.32	−1.21
100	20	0.65	0.33	−1.36	0.41	0.27	−1.32	0.40	0.30	−1.21
100	30	0.55	0.38	−1.37	0.48	0.31	−1.26	0.37	0.22	−1.27
100	40	0.49	0.22	−1.58	0.48	0.27	−1.36	0.39	0.21	−1.20
150	10	0.58	0.48	−2.11	0.34	0.32	−1.81	0.37	0.37	−1.60
150	20	0.44	0.37	−1.87	0.38	0.35	−1.63	0.33	0.34	−1.50
150	30	0.44	0.36	−1.89	0.35	0.31	−1.76	0.39	0.35	−1.58
150	40	0.38	0.37	−2.10	0.36	0.35	−1.73	0.34	0.24	−1.57
200	10	**0.58**	**0.57**	**−2.36**	0.41	0.39	−1.76	0.40	0.43	−1.30
200	20	0.66	0.65	−2.10	0.32	0.37	−1.92	0.22	0.24	−1.74
200	30	0.59	0.68	−2.07	0.31	0.31	−1.93	0.29	0.30	−1.71
200	40	0.49	0.46	−2.24	0.35	0.43	−1.98	0.40	0.44	−1.72
Average		0.55	0.36	−1.54	0.43	0.30	−1.35	0.38	0.27	−1.21

表 1～4 中数据显示,在所有组合上,IVNS 的 ARPD 没有比 H6 和 ILS_RA 差的情况.在大规模算例集上,IVNS 明显好于 H6 和 ILS_RA.在小规模算例集上差别不很明显,因为所有算法都能求得与参考目标值接近的解.比如,H6,ILS_RA 和 IVNS 在小规模算例集上的平均 ARPD 分别 0.18%,0.05%和−0.07%,IVNS 的平均解质量要比 H6 和 ILS_RA 分别好出 0.27%和 0.11%.在大规模算例集上,H6,ILS_RA 和 IVNS 的平均 ARPD 分别为 0.49%,0.33%和−1.34%,IVNS 要比 H6 和 ILS_RA 分别好出 1.86%和 1.66%.图 1 给出了 3 个算法在整个算例集上的 APRD 均值的点图.图 1 显示,IVNS 在整个算例集上亦好于两个参照算法.

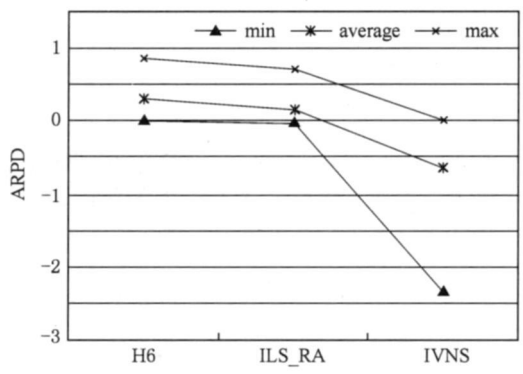

Fig. 1 Average plot for the ARPDs of three tested algorithms.

图 1 三个算法 ARPD 的均值点图

为验证上述结论的客观性,用单因素方差分析对实验数据进行统计分析[23].其中响应变量为每个组合的 ARPD;因素的 3 个水平对应 3 个算法;F 检验的显著概率水平 $\alpha=0.05$.分析结果在表 5 中给出.因 P 值远远小于 α 值,F 值远远大于临界值,F 检验显著,故 3 个算法的 ARPD 均值在统计上差异显著.

Table 5 ANOVA for Experimental Results

表 5 实验结果的单因素方差分析

SUMMARY					ANOVA					
Treatment	Sample Size	Sum	Mean	Variance	Variation Source	Sum of Squares	Freedom Degrees	Mean Squares	F_0	P-value	F Crit
H6	216	68.35	0.32	0.04	Algorithm	112.84	2	56.42	267.18	≪0.01	3.01
ILS_RA	216	36.73	0.17	0.03	Error	136.21	645	0.21			
IVNS	216	−136.7	−0.63	0.56	Total	249.05	647				

图 2 给出了当 $m=40$,加工和准备时间分别为 U[1,100]和 U[0,100]的组合情况下,3 个算法的 ARPD 随工件数的变化趋势.在其他组合下也有类似趋势.图 2 中显示 IVNS 的平均质量随着工件数的增加而更好.由于 CPU 时间设为 $n\times m\times 10$ ms,且 3 个算法的每个组成部分的时间复杂度和机器数呈线性关系,所以 3 个算法对机器数的变化不敏感.

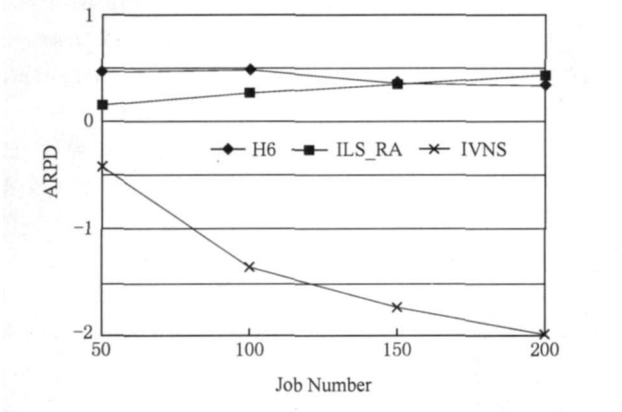

Fig. 2 Trend of ARPD as n increases when $m=40$ and the processing time and setup time from U[1,100] and U[0,100] respectively.

图 2 加工和准备时间分别取自 U[1,100]和 U[0,100]时,ARPD 随工件数增加的变化趋势($m=40$)

综合上述分析,无论从直观还是统计上,IVNS 的寻优性能显著好于 H6 和 ILS_RA.

5 结 论

为求解最小化总完工时间的有准备时间无等待流水车间调度问题,本文提出了 3 个较大规模邻域,分析了它们基本操作的目标增量.较大规模的邻域使局部搜索算法能在更大范围内搜索高质量的解;邻域基本操作的目标增量,使得相邻解的比较可以在常量时间内完成;目标增量将 NEH 插入算法的时间复杂度降低一阶.提出的迭代变化邻域搜索算法 IVNS 通过切换邻域和扰动重启,来克服局部搜索易于限于局部最优的缺陷.通过在大量标准算例上与当前最好算法进行比较,通过对实验结果的统计分析,结果表明 IVNS 优于目前求解所考虑问题的最好算法.

参 考 文 献

[1] Pan Quanke, Zhao Baohua, Qu Yugui. Heuristics for the no-wait flow shop problem with makespan criterion [J]. Chinese Journal of Computers, 2008, 31(7): 1147-1154 (in Chinese)
(潘全科, 赵保华, 屈玉贵. 无等待流水车间调度问题的优化 [J]. 计算机学报, 2008, 31(7): 1147-1154)

[2] Allahverdi A, Ng C T, Cheng T C E, et al. A survey of scheduling problems with setup times or costs [J]. European Journal of Operational Research, 2008, 187(3): 985-1032

[3] Ruiz R, Allahverdi A. Some effective heuristics for no-wait flowshops with setup times to minimize total completion time [J]. Annals of Operations Research, 2007, 156(1): 143-171

[4] Graham R L, Lawler E L, Lenstra J K, et al. Optimization and approximation in deterministic sequencing and scheduling: A survey [J]. Annals of Discrete Mathematics, 1979, 5: 287-326

[5] Aldowaisan T, Allahverdi A. Total flowtime in no-wait flowshops with separated setup times [J]. Computers & Operations Research, 1998, 25(9): 757-765

[6] Allahverdi A. Minimizing mean flowtime in a two-machine flowshop with sequence-independent setup times [J]. Computers & Operations Research, 2000, 27(2): 111-127

[7] Aldowaisan T. A new heuristic and dominance relations for no-wait flowshops with setups [J]. Computers & Operations Research, 2001, 28(6): 563-584

[8] Allahverdi A, Aldowaisan T. No-wait and separate setup three-machine flowshop with total completion time criterion [J]. International Transactions in Operational Research, 2000, 7(3): 245-264

[9] Shyu S J, Lin B M T, Yin P Y. Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time [J]. Computers & Industrial Engineering, 2004, 47(2/3): 181-193

[10] Fischetti M, Laporte G, Martello S. The delivery man problem and cumulative matroids [J]. Operations Research, 1993, 41: 1055-1064

[11] Brown S I, McGarvey R, Ventura J A. Total flowtime and makespan for a no-wait m-machine flowshop with set-up times separated [J]. Journal of the Operational Research Society, 2004, 55(6): 614-621

[12] Li Xiaoping, Wang Qian, Wu Cheng. Heuristic for no-wait flow shops with makespan minimization [J]. International Journal of Production Research, 2008, 46(9): 2519-2530

[13] Li Xiaoping, Wu Cheng. Heuristic for no-wait flow shops with makespan minimization based on total idle-time increments [J]. Science in China Series F: Information Sciences, 2008, 51(7): 896-909

[14] Nawaz M, Enscore Jr E E, Ham I. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem [J]. Omega—International Journal of Management Science, 1983, 11(1): 91-95

[15] Hansen P, Mladenovic N. Variable neighborhood search: Principles and applications [J]. European Journal of Operational Research, 2001, 130(3): 449-467

[16] Stützle T. Iterated local search for the quadratic assignment problem [J]. European Journal of Operational Research, 2006, 174(3): 1519-1539

[17] Schuster C J, Framinan J M. Approximative procedures for no-wait job shop scheduling [J]. Operations Research Letters, 2003, 31(4): 308-318

[18] Feo T, Resende M. Greedy randomized adaptive search procedures [J]. Journal of Global Optimization, 1995, 6(2): 1-27

[19] Fleurent C, Glover F. Improved constructive multistart strategies for the quadratic assignment problem [J]. INFORMS Journal on Computing, 1999, 11(2): 198-204

[20] Taillard ED, Gambardella LM, Gendreau M, et al. Adaptive memory programming: A unified view of metaheuristics [J]. European Journal of Operational Research, 2001, 135(1): 1-16

[21] James T, Rego C, Glover F. Sequential and parallel path-relinking algorithms for the quadratic assignment problem [J]. IEEE Intelligent Systems, 2005, 20(4): 58-65

[22] Rardin R L, Uzsoy R. Experimental evaluation of heuristic optimization algorithms: A tutorial [J]. Journal of Heuristics, 2001, 7(3): 261-304

[23] Montgomery D C. Design and Analysis of Experiments [M]. New York: John Wiley & Sons, 2005

Wang Chuyang, born in 1975. PhD candidate. His main research interests include scheduling algorithms and combinatorial optimization.

王初阳, 1975 年生, 博士研究生, 主要研究方向为调度算法、组合优化.

Li Xiaoping, born in 1970. PhD, associate professor and PhD supervisor. His main research interests include machine scheduling, project scheduling, and service computing.

李小平, 1970 年生, 博士, 副教授, 博士生导师, 主要研究方向为机器调度、项目调度、服务计算.

Wang Qian, born in 1946. Professor and PhD supervisor. Her main research interests include information integration, database technology and CSCW.

王 茜, 1946 年生, 教授, 博士生导师, 主要研究方向为信息集成技术、数据库技术以及计算机协同工作.

Yuan Yingchun, born in 1970. PhD and associate professor. Her main research interests include grid computing and service composition.

苑迎春, 1970 年生, 博士, 副教授, 主要研究方向为网格计算、服务组合.

Research Background

The no-wait flowshop scheduling problem with setup time is considered to minimize the total completion time. In no-wait flow shops, once a job is started on the first machine it has to be continuously processed through the completion on the last machine without interruptions. Applications of no-wait flow shops can be found in many industries such as steel production, food processing, chemical industry, pharmaceutical industry or production of concrete wares. Treating setup time separately from processing time allows operations to be performed simultaneously and hence improved performance. This paper proposes an iterated variable neighborhood search (IVNS) algorithm for the problem considered. Three new neighborhoods are defined to enhance the chance of local search algorithms finding high quality solutions. These neighborhoods are based on job block exchange and have sizes larger than the common used insertion and exchange neighborhoods Objective increments of their moves are developed to speed up the evaluation neighboring solutions. Objective increments are also used to speed up the NEH heuristic. IVNS tries to escape local optima by restarting from perturbations of local optima and varying neighborhoods. Extensive experiments show that IVNS significantly outperforms the reference algorithms. This project is supported by the National Natural Science Foundation of China (No. 60672092, 60504029, 60873236), the National High-Tech Research and Development Plan of China (863-2008AA04Z103) and Natural Science Foundation of Hebei Province (No. F2009000653).