文章编号: 1001-0920(2007)03-0247-05

一类基于 FSP 问题 Block 性质的快速 TS 算法

金 锋, 宋士吉, 吴 澄
(清华大学 自动化系, 北京 100084)

摘 要: 为解决现有算法在求解大规模流水线调度问题(FSP)时计算时间过长的问题,从缩小邻域的角度出发,提出一种基于 FSP 问题 Block 性质的快速禁忌搜索(TS)算法. 利用 Block 性质,算法在保证不丢失最优解的基础上,将邻域中大部分劣解排除,将搜索重点集中在邻域中"最优希望"的区域,以缩小邻域和减少计算时间. 数值仿真实验表明,该算法能在较短时间内获得大规模 FSP 问题的满意解.

关键词: 流水线调度问题; 禁忌搜索; Block 性质
中图分类号: TP301 **文献标识码**: A

Fast TS algorithm based on Block properties of FSP

JIN Feng, SONG Shi-ji, WU Cheng
(Department of Automation, Tsinghua University, Beijing 100084, China. Correspondent: JIN Feng, E-mail: jinfeng99 @tsinghua.org.cn)

Abstract: To the problem that it takes a long time for current available algorithms to solve large-scale flow shop scheduling problems (FSPs), a fast taboo search algorithm based on Block properties of FSP is proposed to reduce the neighborhood size. With the Block properties, most bad solutions in the neighborhood are excluded without losing the optimal solution. The point of search is focused on the "most promising" area to reduce the size of neighborhood and running time. Numerical experiments show that good solutions of large-scale FSPs are found in a short time with the proposed algorithm.

Key words: Flow shop scheduling problem; Taboo search; Block property

1 引 言

流水线调度问题(FSP)是一类典型的组合优化问题,模型描述简单但求解困难(当机器数超过 3 台时,该问题已被证明为 NP 完全问题). FSP 调度问题在工业中有着广泛的应用背景,一直是生产调度领域中的热点之一. 近些年,随着企业生产规模的日益增长,大规模 Flow Shop 调度问题逐渐引起学者的关注.

在当前求解 FSP 问题的算法中,求解中小型规模调度问题(一般不大于 50 个工件,20 台机器)的算法已比较成熟,可大致分为构造型算法和改进型算法两类. 构造型算法包括 Slope Index 算法及其扩展[1,2],CDS 算法[3],RA 算法[4]和 NEH 算法[5]等. 大多数改进型算法是基于元启发式算法的,包括模拟退火[6,7]、禁忌搜索[8-10]和遗传算法[11]等. 构造型算法计算量较少,但得到的解往往不能令人满意,而改进型算法通过迭代可获得较好的解,因而求解较大规模 FSP 问题时,常常先利用构造型算法获取初始解,然后利用改进型算法进行迭代改进. 但随着调度问题规模的增长,改进型算法为获得满意解所需的计算量迅速增加,当规模增大到一定程度时,不少算法因运行时间过长而变得不可行. 因此,求解大规模 FSP 问题的关键在于在保证算法寻优性能的基础上,降低算法的计算量,从而减少计算时间.

大多数改进型算法是基于邻域搜索的,因此,在一定的邻域结构下,缩小搜索邻域使算法计算量减少是常用的方法. 如 Zegordi 等[12]通过引入 MDJ(工件移动需求度)来缩小邻域; Nowicki 等[8]通过分析调度方案内部的 Block 结构,然后引入控制因子 $\varepsilon \in [0,1]$,使搜索邻域缩小为原邻域的 ε 倍. 但盲目缩小邻域会使得邻域中某些更好的解丢失,从而使得算法寻优性能下降. Grabowski 等[9,10]在文献

收稿日期: 2005-11-18; **修回日期**: 2006-01-04.
基金项目: 国家 973 重点基础研究项目(2002CB312205); 国家自然科学基金项目(60574077).
作者简介: 金锋(1981—),男,江苏无锡人,博士生,从事大规模生产调度的研究; 吴澄(1940—),男,浙江桐乡人,中国工程院院士,教授,博士生导师,从事复杂生产系统的排序、调度和可靠性研究.

[8]的基础上,进一步研究 Flow Shop 问题的 Block 性质,由此极大拓展了缩小邻域的思路.

本文在文献[9, 10]的基础上,将 FSP 调度问题的 Block 性质引入禁忌搜索(TS)算法中,利用 Block 性质在 $O(1)$ 时间内获取邻域中其他解的下界,从而将大部分"劣解"(即下界比已知上界还大的解)忽略,只搜索那些"最有希望"的区域. 由此既保证了最优解不丢失,又使得搜索邻域大为缩小,在保证解质量的基础上降低了算法计算量,为求解大规模 FSP 问题提供了可能.

2 问题描述和预备知识

2.1 FSP 问题模型

FSP 问题研究的是 n 个工件在 m 台机器上的加工过程,用 p_{ij} 表示第 i 个工件在第 j 台机器上的加工时间. FSP 问题要求不同工件在各个机器上加工顺序相同. 问题的目标是在满足加工顺序的前提下,确定各个工件在各台机器上的加工时间和加工顺序,使得某些性能指标达到最优.

FSP 问题的解可用排列 $\pi = (\pi(1), \cdots, \pi(n))$ 表示,其中 $\pi(1), \cdots, \pi(n)$ 是工件号,其先后顺序即为工件在机器上的加工顺序. 用 Π 表示这些排列的集合,希望能找到一个最优排列 $\pi^* \in \Pi$,使得

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi), \tag{1}$$

其中 $C_{\max}(\pi)$ 是以 π 为顺序进行加工所需的总工期.

在已有文献中,总工期 $C_{\max}(\pi)$ 还可表述为

$$C_{\max}(\pi) = \max_{1 \le t_1 \le t_2 \le \cdots \le t_{m-1} \le n} \left(\sum_{j=1}^{t_1} p_{\pi(j)1} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m} \right). \tag{2}$$

式(2)的来源和直观含义参见附录.

2.2 关键路径和 Block 性质

在式(2)的基础上,可定义解 π 的关键路径和 Block 结构.

定义 1[9] 称 $u = (u_1, u_2, \cdots, u_{m-1})$ 为关键路径,假如 u 满足

$$C_{\max}(\pi) = \sum_{j=1}^{u_1} p_{\pi(j)1} + \sum_{j=u_1}^{u_2} p_{\pi(j)2} + \cdots + \sum_{j=u_{m-1}}^{n} p_{\pi(j)m}. \tag{3}$$

定义 2[9] 对于关键路径 u,称工件序列 $B_k = (\pi(u_{k-1}), \pi(u_{k-1}+1), \cdots, \pi(u_k))$ 为该关键路径的第 k 个 Block,并定义其内部 Block 为

$$B_k^* = \begin{cases} B_k - \{\pi(u_1)\}, & k = 1; \\ B_k - \{\pi(u_{k-1}), \pi(u_k)\}, & 1 < k < m; \\ B_k - \{\pi(u_{m-1})\}, & k = m. \end{cases} \tag{4}$$

下面通过例子对上述标记和概念加以阐述.

例 1 图 1 给出了 FSP 问题(含 7 个工件,3 台机器)的一个调度方案. 该方案相应的解为排列 $\pi = (1,7,3,2,4,5,6)$,其关键路径为 $u = (2,6)$,该关键路径产生了 3 个 Block,即 $B_1 = (1,7)$, $B_2 = (7,3,2,4,5)$ 和 $B_3 = (5,6)$ 以及 3 个内部建筑块 $B_1^* = (1)$, $B_2^* = (3,2,4)$ 和 $B_3^* = (6)$.

2.3 邻 域

FSP 问题解的邻域往往是通过将解 π 中两个或多个工件变换位置产生的. 常用的方法有交换和插入两种操作. 已有不少学者通过实验表明[13],利用插入操作形成的邻域要比交换操作更有利于算法寻优. 因此,本文采用插入操作形成的邻域.

用 $v = (a,b)\ (a,b \in \{1, \cdots, n\}, a \neq b)$ 表示对排列 π 中的一个插入操作,即将排列 π 中位置 a 的工件取出,插入到位置 b. 由此可得到一个新的排列

$$\pi_v = \begin{cases} (\pi(1), \cdots \pi(a-1), \pi(a+1) \cdots \pi(b), \\ \pi(a)\pi(b+1), \cdots, \pi(n)), & a < b; \\ (\pi(1), \cdots \pi(b-1), \pi(a)\pi(b+1) \cdots \\ \pi(a-1), \pi(a+1), \cdots, \pi(n)), & a > b. \end{cases}$$

根据此排列 π,由插入操作形成的邻域可定义为

$$N(\pi) = \{\pi_v \mid v \in V\},$$

其中 $V = \{(a,b) \mid b \notin \{a-1, a\}, a, b, \in \{1,2, \cdots, n\}\}$.

从邻域的定义可见,插入操作形成的"原始"邻域内有 $(n-1)^2$ 个元素. 为获取邻域中的最好解,需要对邻域中的每一个解进行评估(即计算该解的总工期). 而评估一个解所需时间复杂度为 $O(mn)$,因此,若对"原始"邻域内所有解都评估,则需要时间为 $O(mn^3)$,而这仅仅是评估一个解的邻域所需时间. 在改进型算法的迭代过程中,往往需要评估成千上万个解的邻域,因此,随着调度问题规模增大,尤其是工件数增加时,评估所需时间迅速增加. 为此,在 2.2 节 Block 概念的基础上,引入 FSP 问题的 Block 性质,以减少搜索邻域的大小.

定理 1[9] 对任意排列 $\pi \in \Pi$,若将 π 中第 k 个 Block 内的元素 $\pi(a)$ 移动到第 l 个 Block 内的位置

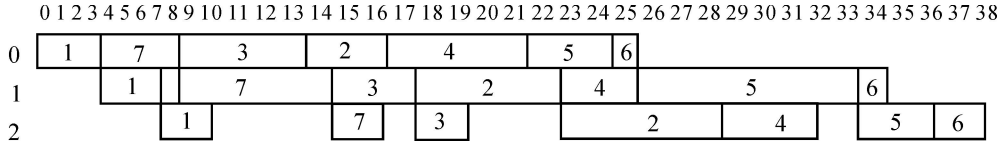

图 1 排列 $\pi = (1,7,3,2,4,5,6)$ 的甘特图

b,那么由操作 $v=(a,b)$ 所得 π_v 的总工期满足

$$C_{\max}(\pi_v) \geq C_{\max}(\pi) + p_{\pi(a)l} - p_{\pi(a)k}. \quad (5)$$

$p_{\pi(a)l}$ 和 $p_{\pi(a)k}$ 是工件 $\pi(a)$ 在第 l 台机器和第 k 台机器上的加工时间,是已知的,因此,在已知 $C_{\max}(\pi)$ 的情况下,由定理 1 能在 $O(1)$ 时间内获得 $C_{\max}(\pi_v)$ 的一个下界,不妨记为 $\mathrm{LB}(\pi_v)=C_{\max}(\pi)+p_{\pi(a)l}-p_{\pi(a)k}$.

显然,若 $p_{\pi(a)l} \geq p_{\pi(a)k}$,则由定理 1 可以得到 $C_{\max}(\pi_v) \geq C_{\max}(\pi)$,那么 π_v 不用评估便可知道其性能不比会 π 好,若把这样的解忽略,则不会影响算法在 π 的邻域内找到比 π 更好的解.

在此基础上,可将原始邻域缩小,定义如下搜索邻域:

$$N(\pi,\mathrm{UB}) = N(\pi) - \{\pi_v \mid \mathrm{LB}(\pi_v) \geq \mathrm{UB}\}. \quad (6)$$

其中:$\mathrm{LB}(\pi_v)$ 是根据式(5)计算所得的 $C_{\max}(\pi_v)$ 一个下界,UB 是设定的一个上界. 式(6)意味着将邻域中性能比 UB 差的一些解排除掉.

若令 $\mathrm{UB}=C_{\max}(\pi)$,那么在邻域 $N(\pi,\mathrm{UB})$ 中搜索时,不会丢失总工期比 π 更优的解. 进一步,若令 $\mathrm{UB}=C_{\max}(\pi^*_{\mathrm{cur}})$,其中 π^*_{cur} 是搜索过程中已知的最好解(一般要比某次迭代中的基解 π 好),那么搜索过程中不会丢失比 π^*_{cur} 更优的解.

对于 $N(\pi,\mathrm{UB})$,若令 $\mathrm{UB}=C_{\max}(\pi)$,则以下两点是值得注意的:

1) 注意到在式(5)中并没有出现移动后的位置 b,这意味着若有 $p_{\pi(a)l} \geq p_{\pi(a)k}$,那么将工件 $\pi(a)$ 移动到 Block l 中任何位置产生的解,均可被忽略. 这在 $n \gg m$ 情形下(平均每个 Block 大小为 n/m),被忽略的解是非常可观的.

2) π 性能越差,则其搜索邻域 $N(\pi,\mathrm{UB})$ 越小. 因为 π 性能越差,意味着 $C_{\max}(\pi)$ 越大,则由式(5)可得 $\mathrm{LB}(\pi_v)$ 也越大,该解被排除的可能也就越大. 在 $N(\pi)$ 大小固定的条件下,$N(\pi,\mathrm{UB})$ 就越小.

利用 Block 性质还可以进一步缩小搜索邻域. 假设已通过操作 $v=(a,b)$ 获取到 π 邻域内的解 π_v,那么对于操作 $v'=(a,c)$ 产生的解 $\pi_{v'}$,既可以看成 π 邻域内的元素,又能看成 π_v 邻域内的元素. 因此既能由 π 的 Block 结构获取 $\pi_{v'}$ 的下界 $\mathrm{LB}(\pi_{v'})$,又能进一步分析 π_v 的 Block 结构获取下界 $\mathrm{LB}'(\pi_{v'})$. 这里取 $\mathrm{LB}(\pi_{v'})=\max\{\mathrm{LB}(\pi_{v'}),\mathrm{LB}'(\pi_{v'})\}$ 作为 $\pi_{v'}$ 新的下界,显然新的下界不会比原下界差. 根据式(6),被排除的邻域元素也就更多,搜索邻域进一步被缩小.

例 2　仍以例 1 中的问题为例. 考虑把 $\pi=(1,7,3,2,4,5,6)$ 中的工件 1 取出插入到其他位置形成的邻域. 由于工件 1 在第 1 个 Block 内部,且 $p_{12} > p_{11} > p_{13}$,由定理 1 可知,插入到第 2 个 Block 内形成的 4 个解 $(7,1,3,2,4,5,6)$,$(7,3,1,2,4,5,6)$,$(7,3,2,1,4,5,6)$ 和 $(7,3,2,4,1,5,6)$ 不会比 π 好,因此不必再评估,而只用评估插入到第 3 个 Block 内的两个解 $(7,3,2,4,5,1,6)$ 和 $(7,3,2,4,5,6,1)$(而这两个解也可利用定理 1 得到一个下界).

3　基于 Block 性质的 TS 算法

禁忌搜索(TS)算法是求解组合优化问题常用的智能优化算法之一[14],在生产调度邻域得到了广泛的应用. 其基本思路是,从一个初始解出发,找出其邻域中满足一定条件的性能最好的解,把该解作为新的基础解,再搜索其邻域,由此不断迭代直到满足停止条件. 由一个基础解到下一个基础解称为一次移动. 禁忌搜索算法通过禁忌表禁止某些移动,以避免重复搜索或从局部最小点跳出.

TS 算法的基本元素有:初始解、禁忌表和禁忌状态、邻域和搜索策略. 其中邻域的选择对算法的寻优性能和效率都有很大的影响,本文提出的算法将搜索限制在邻域中"最有希望"的区域,既保证不丢失最优解,又使得搜索邻域大大缩小. 下面对算法中的各个元素分别加以阐述.

3.1　初始解

在本文算法中,以 NEH 算法构造的解作为初始解. 原始的 NEH 算法复杂度是 $O(mn^3)$,本文采用 Taillard 等[13]的实现技巧,使 NEH 算法的复杂度降为 $O(mn^2)$.

3.2　禁忌表和禁忌状态

用 $T=(T_1,\cdots,T_{\mathrm{maxt}})$ 表示一个禁忌表,其中 maxt 是禁忌表长,$T_i=(g,h)$ 是工件对. 假设排列 π 经过某个插入操作 $v=(a,b)$ 移动到排列 π_v,若 $a<b$,则将工件对 $(\pi(a),\pi(a+1))$ 加入到禁忌表中;否则将 $(\pi(a-1),\pi(a))$ 加入禁忌表. 如果禁忌列表长度超过 maxt,则将最早加入禁忌表的元素从禁忌表中移去.

在搜索过程中,符合下面条件之一的移动 $v=(a,b)$ 是被"禁忌"的:

1) 若 $a<b$,有一个或多个工件号对 $(\pi(j),\pi(a))$,$j=a+1,\cdots,b$ 在禁忌表中;

2) 若 $a>b$,有一个或多个工件号对 $(\pi(a),\pi(j))$,$j=b,\cdots,a-1$ 在禁忌表中.

3.3　邻域和搜索策略

根据 2.3 节的分析,本文设计了 3 种邻域,并分别在这 3 种邻域下进行搜索. 这 3 种邻域分别是:

1) 原始邻域 $N(\pi)$. 对于任意基础解 π,该邻域中均有 $(n-1)^2$ 个元素.

2) 经过分析基础解 π 的 Block 结构后,根据式(6)得到邻域 $N(\pi,\mathrm{UB})$. 在算法中,取 $UB=$

$C_{\max}(\pi^*_{\text{cur}})$,其中 π^*_{cur} 为搜索过程中已知的最好解.

3) 在 $N(\pi, C_{\max}(\pi^*_{\text{cur}}))$ 的基础上,进一步分析邻域元素的 Block 结构,将 2) 中的邻域进一步缩小.

这里将运用邻域 1) 的 TS 算法称为 TS_All,运用邻域 2) 和邻域 3) 的 TS 算法分别称为 TS_Block1 和 TS_Block2.

在邻域搜索过程中,一旦找到比已知最好解更好的解,则取该解为新的基础解,否则取搜索邻域中性能最好的解(到达该解的操作未被禁忌) 作为新的基础解.

3.4 算法总体流程

3 种 TS 算法(TS_All,TS_Block1,TS_Block2) 在流程上相差不大,下面对最复杂的 TS_Block2 算法进行简单阐述.

输入:基础解 π_{basic} = Null,已知最好解 π^*_{cur} = Null,最大迭代次数 maxIter,禁忌列表长度 maxt,工件加工时间 p_{ij},当前邻域已知最好解 $\pi^*_{\text{cur}N}$ = Null.

输出:最大迭代次数内找到的最好解 π^*_{cur}.

Step1:用 NEH 算法获取初始解 π_0,令 $\pi_{\text{basic}} = \pi_0$,$\pi^*_{\text{cur}} = \pi_0$.

Step2:分析 π_{basic} 的Block结构,运用式(5) 计算邻域内元素下界,缩小邻域.

Step3:对搜索邻域中未被禁忌的解 π_v,按如下进行:

Step3.1:评估 π_v,获得 $C_{\max}(\pi_v)$;

Step3.2:若 $C_{\max}(\pi_v) < C_{\max}(\pi^*_{\text{cur}})$,则令 $\pi^*_{\text{cur}} = \pi_v$,$\pi_{\text{basic}} = \pi_v$,更新禁忌表,返回 Step2;

Step3.3:若 $C_{\max}(\pi_v) < C_{\max}(\pi^*_{\text{cur}N})$,则令 $\pi^*_{\text{cur}N} = \pi_v$;

Step3.4:分析 π_v 的Block结构,更新邻域下界,缩小邻域.

Step4:是否已到最大迭代次数,若到,则中止算法;否则令 $\pi_{\text{basic}} = \pi^*_{\text{cur}N}$,返回 Step2.

对 TS_All 算法,Step2 和 Step3.4 不用执行,对算法 TS_Block1,Step3.4 不用执行,其他步骤都一样.

3.5 算法复杂度分析

在 3.4 节的各个步骤中,Step1 中 NEH 算法的复杂度为 $O(mn^2)$;Step2 中分析 π_{basic} 的 Block 结构的复杂度为 $O(m+n)$;Step3 是最费时间的,这里采用文献[8] 的实现技巧,评估邻域中所有元素(即 TS_All 算法) 所需的时间为 $O(mn^2)$. TS_Block1 算法和 TS_Block2 算法的计算量与问题的 Block 分布有关,其算法复杂度难以精确确定,但肯定不高于 $O(mn^2)$. 因此,若将最大迭代次数 maxIter 视为常数,则 3 种算法的复杂度均不超过 $O(mn^2)$.

4 仿真结果

本文用 VC++ 实现了 TS_All 算法,TS_Block1 算法以及 TS_Block2 算法,并在 P4(2.6 GHz),512 M 内存的计算机上进行测试. 所有算法都用到 Taillard[15] 中规模最大的 50 个 FSP 问题 Ta71 ~ Ta120,规模分别为 100 ×10,100 ×20,200 ×10,200 ×20,500 ×20,每个规模内包含 10 个问题,并重点测试了 500 ×20 规模下的 10 个问题. 为便于比较,3 个算法禁忌表长度均取为 8,最大迭代次数取为 1 000.

图 2 为 3 种算法求解中大规模 FSP 问题所需的平均 CPU 时间,图 3 为 3 种算法求解现有文献中最大规模(500 ×20) 的 10 个 FSP 问题所需 CPU 时间. 从图 2 中可以看出,随着问题规模的增长,TS_All 算法所需时间迅速增长,TS_Block1 所需 CPU 时间比 TS_Block2 算法略多,但总体上后两种算法增长速度要比 TS_All 算法慢得多. 从图 3 可以看出,求解 500 ×20 这样的大规模 FSP 问题,后两种算法所需时间只需 TS_All 算法的一半. 这是因为 TS_All 搜索了邻域中所有元素,而后两个算法只搜索邻域中"最有希望"的部分,降低了计算量,从而减少了计算时间.

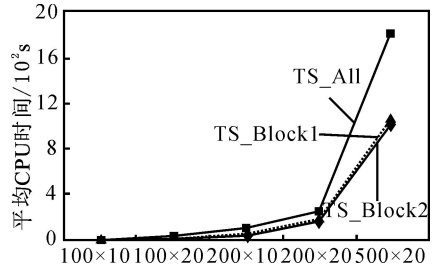

图 2 各算法求解不同规模所需平均 CPU时间

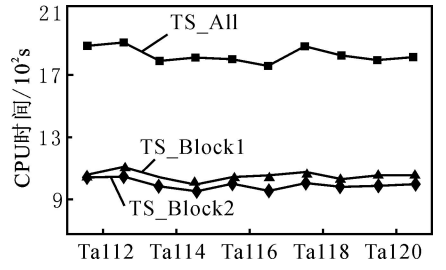

图 3 各算法求解 500 ×20 规模问题所需 CPU时间

在已有文献中,文献[8-10] 中的算法在求解大规模 FSP 时寻优性能最好,所得结果也最好. 综合这 3 篇文献中的结果,取最好解记为 $C^*_{\max}$. 图 4 给出了 3 种算法在最大迭代次数内找到的最好解 $C^H_{\max}$(H 代表不同算法) 与 $C^*_{\max}$ 之间的相对偏差 $\text{PRD}(H) = (C^H_{\max} - C^*_{\max})/C^*_{\max}$. 从图中可以看出,

TS_All 算法获得的解的性能与已知最好解的偏差较大,而 TS_Block1 与 TS_Block2 算法获得的解与已知最好解相差不多(最大偏差约为 0.5 %),在一些问题上(如 Ta114 ～ Ta117)还获得了比文献[8-10]更好的解.

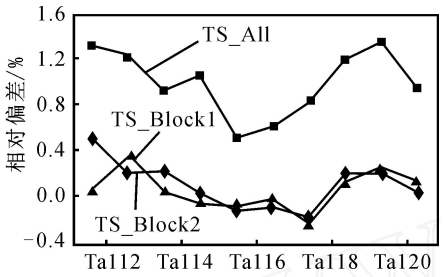

图 4　各算法所得解与已知最好解的相对偏差

综合以上结果可以看出,基于 Block 性质的 TS 算法(TS_Block1 算法和 TS_Block2 算法)缩小了搜索的邻域,因而运算时间大为减少. 此外,算法搜索时始终集中于邻域中"最有希望"的区域,从而使算法的优化性能得到了提高.

5　结　　语

本文提出了一种基于 FSP 问题的 Block 性质的快速 TS 算法. 算法从缩小邻域的角度出发,利用 FSP 的 Block 性质,获取邻域中元素的下界,以此将算法搜索范围限制在邻域中"最有希望"的区域,从而有效减小了算法的计算量,提高了算法寻优能力,为求解大规模 FSP 问题提供了可能. 数值仿真实验表明,用本文算法对中大规模 FSP 问题进行求解,可在较短时间内获得问题的近优解.

基于本文结果,可进一步研究 FSP 问题的结构性质,并将这些性质引入算法以提高算法的效率和寻优性能.

附　　录

在已有文献中,Flow Shop 的解 π 通常也用图 $G(\pi) = \{N, E\}$ 表示(如图 5 所示),其中 N 表示节点集合,节点 (i, j) 的权重为 $p_{\pi(j)i}$, E 表示边集. 显然 $C_{\max}(\pi)$ 即为节点 $(1,1)$ 到节点 (m,n) 的最长加权路径. 而从节点 $(1,1)$ 到节点 (m,n) 的任一路径均可用序列 $t = (t_1, \cdots, t_{m-1})$, $0 \leq t_1 \leq \cdots \leq t_{m-1} \leq n$ 表示,由 t 可将整个路径划分为 m 个"水平子路径",在第 i 台机器上的"水平子路径"包含节点 (i, t_{i-1}), $(i, t_{i-1}+1)$, $\cdots$, (i, t_i),其加权长度和为 $\sum_{j=t_{i-1}}^{t_i} p_{\pi(j)i}$,由此路径 t 的全部加权长度和可表示为

$$C(\pi, t) = \sum_{j=1}^{t_1} p_{\pi(j)1} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m}.$$

由于 $C_{\max}(\pi)$ 为节点 $(1,1)$ 到节点 (m,n) 的最长加权路径,$C_{\max}(\pi)$ 可表示为

$$C_{\max}(\pi) = \max_{1 \leq t_1 \leq t_2 \leq \cdots \leq t_{m-1} \leq n} \left(\sum_{j=1}^{t_1} p_{\pi(j)1} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m} \right),$$

即为原文中的式(2).

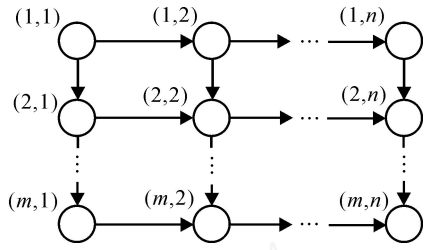

图 5　排列 π 的网络图 G(π)

参考文献(References)

[1] Gupta J. A functional heuristic algorithm for the flowshop scheduling problem[J]. Operational Research Quarterly, 1971, 22(1): 39-47.

[2] Palmer D. Sequencing jobs through a multistage process in the minimum total time: A quick method of obtaining a near optimum[J]. Operational Research Quarterly, 1965, 16(1): 101-107.

[3] 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): B630- B637.

[4] Dannenbring D G. An evaluation of flow shop sequencing heuristics[J]. Management Science, 1977, 23(11): 1174- 1182.

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

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

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

[8] 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.

[9] Grabowski J, Pempera J. New block properties for the permutation flow shop problem with application in tabu search[J]. J of the Operational Research Society, 2001, 52(2): 210-220.

[10] Grabowski J, Wodecki M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion [J]. Computers and Operations Research, 2004, 31(11): 1891-1909.

(下转第 257 页)

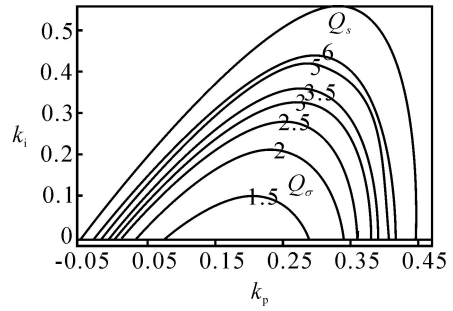

图 3　当 k_d^* = 0.1 时 k_p - k_i 平面上的等 σ^2 线

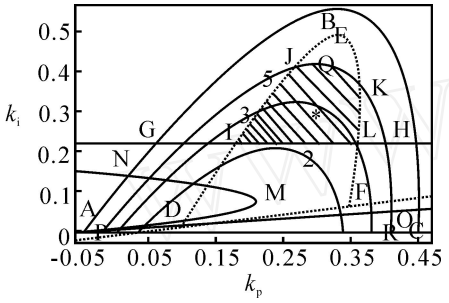

图 4　对应 k_d^* = 0.1 时相容性解集 Q_c

闭区域为 $Q_{c_1}(k_d^*)$, 由 AMNBO 围成的闭区域为 $Q_{c_2}(k_d^*)$, 由 PQR 围成的闭区域为 $Q_\sigma(k_d^*)$. 取一定步长如 h = 0.1 扫描 Q_s 内的 k_d 值区间, 重复上述工作, 就可以求得相容性解集 Q_c, 对 $\forall q \in Q_c$, 式 (5) ～ (7) 同时成立. 取 q^* = [0.3　0.3　0.1] ∈ Q_c (k_d^* = 0.1) (图 4 中标 * 号的点), 对应的 Re(λ) = - 0.409 1, c_0 = 0, c_1 = 0.143 5, c_2 = 0.820 2, σ_y^2 = 2.905 9, 满足式 (5) ～ (7).

5　结　　语

在工程实践中, 所设计的控制器通常要满足多个性能指标, 人们所期望的控制策略应是相容区域尽可能大的满意解集, 这样为控制器进一步设计和物理实现留有更大的自由度. 本文对一类随机系统的 PID 控制所涉及的上述问题, 给出了一种有效的解决方法.

参考文献(References)

[1] 王远钢, 郭治. 状态反馈中圆形极点和状态方差约束的相容性[J]. 自动化学报, 2001, 27(2): 207-213.
(Wang Y G, Guo Z. Consistency of circular pole and state variance constraints in state-feedback control [J]. Acta Automatica Sinica, 2001, 27(2): 207-213.)

[2] 程相权, 郭治, 王远钢. 满足 H_∞ 区域极点和方差指标约束的动态输出反馈控制研究[J]. 控制与决策, 2002, 17(3): 282-286.
(Cheng X Q, Guo Z, Wang Y G. On dynamic output control with constraints on H_∞, pole placement and covariance[J]. Control and Decision, 2002, 17(3): 282-286.)

[3] 王远钢, 郭治. 反馈控制系统多性能约束指标的相容性[J]. 控制理论与应用, 2003, 20(3): 423-426.
(Wang Y G, Guo Z. Consistency of multiple performance indices of feedback control systems [J]. Control Theory and Applications, 2003, 20(3): 423-426.)

[4] Guo Z. A Survey of satisfying control and estimation [C]. Proc of the 14th IFAC World Congress. Beijing, 1999: 443-447.

[5] Åström K J, Hägglund T. PID controllers: Theory, design, and tuning [M]. NC: Instrument Society of American, 1995.

[6] Bhattacharyya S P, Chapellat H, Keel L H. Robust control: The parametric approach [M], NJ: Prentice Hall PTR, 1995.

[7] 胡寿松. 自动控制原理[M]. 北京: 国防工业出版社, 2000.
(Hu S S. Principles of automatic control [M]. Beijing: National Deference Industry Press, 2000.)

[8] Skelton R E, Iwasaki T. Lyapunov and covariance controllers[J]. Int J Control, 1993, 57(3): 519-536.

[9] Laurent E G, Silviu-Iulian N. Advances in linear matrix inequality methods in control [M]. Philadelphia: SIAM Society for Industrial and Applied Mathematics, 1999.

[10] Jury E I, Dewey A G. A general formulation of the total square integral for continuous systems[J]. IEEE Trans on Automatic Control, 1965, 10(1): 119-120.

(上接第 251 页)

[11] Reeves C R, Yamada T. Genetic algorithms, path relinking, and the flowshop sequencing problem [J]. Evolutionary Computation, 1998, 6(1): 45-60.

[12] Zegordi S H, Itoh K, Enkawa T. Minimizing makespan for flow shop scheduling by combining simulated annealing with sequencing knowledge [J]. European J of Operational Research, 1995, 85(3): 515-531.

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

[14] 贾永基, 谷寒雨, 席裕庚. 求解 PDPTW 问题的一种快速禁忌搜索算法[J]. 控制与决策, 2004, 19(1): 57-60.
(Jia Y J, Gu H Y, Xi Y G. Quick taboo search algorithm for solving PDPTW problem[J]. Control and Decision, 2004, 19(1): 57-60.)

[15] Taillard E. Benchmarks for basic scheduling problems [J]. European J of Operational Research, 1993, 64(2): 278-285.