第34卷 第16期 计 算 机 工 程 2008年8月
Vol.34 No.16 Computer Engineering August 2008
·博士论文· 文章编号：1000—3428(2008)16—0019—03 文献标识码：A 中图分类号：TP391

基于动态规划的资源受限随机工序调度

蒋 维[1]，陈 开[1]，钟小强[1]，王成恩[2]，竺长安[1]
(1. 中国科学技术大学工程科学学院，合肥 230027；2. 东北大学教育部流程工业重点实验室，沈阳 110004)

摘 要：为解决资源受限条件下的随机工序调度问题，该文提出一种基于离散随机动态系统描述的加工时间离散随机分布且同时具有不兼容和多种可更新资源约束的资源受限项目调度模型，使得在满足资源约束和工序约束的前提下，总的平均加工时间最短。该系统研究了动态规划算法求解该问题的方法。通过实例，验证了该方法的有效性和可行性。

关键词：资源受限；随机工序调度；动态规划

Resource Constrained Stochastic Job Scheduling Based on Dynamic Programming

JIANG Wei[1], CHEN Kai[1], ZHONG Xiao-qiang[1], WANG Cheng-en[2], ZHU Chang-an[1]
(1. School of Engineering Science, University of Science and Technology of China, Hefei 230027;
2. Key Lab of Process Industry Automation of Ministry of Education, Northeastern University, Shenyang 110004)

【Abstract】To deal with resource constrained stochastic job scheduling problem, a discrete-time job during distribution stochastic project scheduling problem subject to an incompatibility constraint and multiple renewable resource constraints is described by a discrete-time discrete-event dynamic system. The objective is to minimize the expected project duration under the resource and job order constraints. The optimal solution can be obtained by solving a stochastic dynamic program. The solution modeled as a Markov decision process is described in detail. A typical example validates the feasibility of the method.

【Key words】resource constrained; stochastic job scheduling; dynamic programming

1 概述

资源受限项目调度(Resource Constrained Project Scheduling Problem, RCPSP)是一类广义的问题，要求在满足项目紧前约束与资源约束的前提下，调度项目所有任务的开工期和完工期与最小化项目总工期和最大化资源利用率。它在新产品的开发、单件全订货型生产的规划与调度管理、基建项目施工调度等领域有着广泛的应用。理论上该问题属于NP-hard 问题，其模型丰富，许多组合优化问题如 Job Shop, Flow Shop，单机与并行多机等的调度问题都是它的特殊情形。因此，研究 RCPSP 具有重要的理论和现实意义。

工艺规划和调度是计算机集成制造系统的重要组成部分，在这 2 个子系统中，都包括了资源的分配。其中，资源受限的工序调度问题就成为了研究的热点之一[1-4]。与前人研究的问题有所不同，本文考虑了不兼容的约束和多种加工模式，其中每种模式与一系列确定的资源需求和离散的加工时间分布相关联，提出一种用离散随机动态系统描述的基于加工时间离散随机分布的RCPSP模型，并给出了一个动态规划算法，来对生产过程进行模拟，使得在满足资源约束和工序约束的前提下，总的平均加工时间最短。

2 RCPSP 的马尔可夫过程模型

典型的 RCPSP 主要包括：已知的工序集合和有限的资源集，目标则是求出工序加工序列，使得目标函数取得极值，在本文中为最小化加工时间。将约束分为 2 个部分：前置约束和资源约束。前置约束是指该工序必须在约束中所有的工序完成后才能开始；资源约束是指在同一时刻所有进行的工序的资源总和要小于整个项目所能供给的数目。

加工时间有确定性和随机性之分。本文采用随机工序。考虑工序间存在的不兼容性，加入不兼容约束。建立一工序模型如下：

工序号

约束(前置约束，不兼容约束)+随机分布的加工模式(加工模式集) (1)

其中，加工模式集为资源需求+时间概率分布。

每一个工序可以有多种模式进行加工，每一模式都要求一定量的资源。在式(1)中的加工模式集的每一元素都有资源需求和时间概率分布。时间概率分布为马尔可夫链[5]，时间间隔为 1，并从时刻 0 开始取值。如一马尔可夫链为(0.3, 0.4, 0.4)，表示工序在 0 时刻完成的概率为 0.3，在 1, 2 时刻完成的概率都为 0.4。对该工程，有下列假设：

(1)工序在某一模式下的加工时间为随机分布，但各个模式的随机分布是确定的。

(2)1 个工序只有在它所有的前置工序都完成后才能开始。

(3)1 个工序开始进行后不能被中途中断。

(4)所有的资源都是可更新的，并且不考虑损耗。

(5)2 个工序之间没有时间间隔。

基金项目：中国科学院创新基金资助项目(200417009)
作者简介：蒋 维(1979－)，男，博士研究生，主研方向：生产过程建模与控制，制造系统集成；陈 开、钟小强，博士研究生；王成恩、竺长安，教授、博士、博士生导师
收稿日期：2007-09-20 **E-mail：**wetting@mail.ustc.edu.cn

2.1 定义

对每一工序，定义 3 种状态：等待，处理和完成：

$$j:\ \text{工序状态},\quad j\in\{-1,i,M\} \tag{2}$$

其中，-1 表示等待；i 表示工序在处理过程中采用的模式；M 表示工序完成状态，设定比模式数大一个数量级为(99)。系统的状态可以描述为

$$S=\{j_1,j_2,\cdots,j_i,\cdots,j_n\} \tag{3}$$

其中，$i\in[1,n]$，n 为工序号。在 S 末尾，附加一时间序列，表示系统中工序离完成还需的时间。

$$T=\{T_1,T_2,\cdots,T_i,\cdots,T_n\}\ \ i\in[1,n] \tag{4}$$

整个系统的状态描述为：$\{S(n),T(n)\}$，该状态为马尔可夫链。

当前状态为

$$St(n)=\{S(n);T(n)\} \tag{5}$$

从状态 (x,α) 到 (y,β) 条件转移概率为

$$P(x,\alpha;y,\beta)=P(S(n+1)=y,T(n+1)=\beta\mid S(n)=x,T(n)=\alpha) \tag{6}$$

$$S_a=\{j_{a1},j_{a2},\cdots,j_{am}\}\quad j_{ai}\in S\ ,m\le n \tag{7}$$

式(7)为式(3)中的处于处理状态的项组成的集合。

$$P(x,\alpha,y,\beta)=\prod_{i=1}^{m}\prod_{j=0}^{n}job[i].dist[j] \tag{8}$$

其中，$job[i]$ 表示从 x 到 y 状态，处理状态集式(7)中第 i 项的序号；$dist[j]$ 表示该模式下的随机时间模式。

当前各工序的剩余时间，与下一状态时间的关系如下：

$$T_{re}(x,\alpha)=Val(x,\alpha)+\sum_{i=1}^{p}P(x,\alpha,y_i,\beta_i)T_{re}(y_i,\beta_i) \tag{9}$$

其中，$T_{re}(x,\alpha)$, $Val(x,\alpha)$ 分别是系统在状态 (x,α) 下的剩余时间和有效时间。$Val(x,\alpha)$ 取 $\{\alpha\}$ 中当前处理的工序号中对应的最小值；p 是从当前状态可能进入的下一状态的数值；$P(x,\alpha,y_i,\beta_i)$ 是从当前状态 (x,α) 到下一状态 $\{y_i,\beta_i\}$ 的转移概率。对于终止状态，显然有 $T_{re}=0$。从最终状态逆求出各状态时间。

2.2 基本模型

模型的基础是工序集。每一工序采用式(1)的形式。该工序的剩余时间和后置工序集，需要结合其他的工序来计算。在工程中还应包括总资源数，并指定最后加工的工序。

工程：

总资源数目

工序集 {1; 2; 3; …}

最后加工的工序。

3 算法描述

定义 3 种数据结构 *SyS*, *SySNode* 和 *Trans*：

```
class SyS{              class SySNode{      class Trans{
SySNode *head;        JobArray arJ;       TimeArray arT;
SySNode *current;     int artLength;      double value;
SySNode *tail:};      Trans* arT;          JobArray arJ;};
                        SySNode *next;};
```

JobArray 和 *TimeArray* 分别代表式(3)、式(4)中各工序的状态和剩余时间。对于每一个工序状态集 *JobArray*，有多种可能的下一状态，定义一动态数组来存储这些后续的状态；*SySNode* 表示某一状态的 *JobArray* 的特征；*artLength* 表示该状态可能的下一状态的个数；*Trans* 存储当前状态到下一状态的转移特征，包括各工序剩余时间、新的状态特征以及新状态的平均完成时间特征；*SyS* 用于将已经计算的最优状态转移集链接起来，遍历该链表即可获得工程的最优特征。

3.1 主算法

Step1(初始化) 输入和初步处理。由工序数获得 *JobArray* 和 *TimeArray* 中元素个数。*SySNode* 中的 *arJ* 设为(0,0,⋯,0)，*length* 为 1，*TimeArray* 为(-1,-1,⋯-1)，*value* 为 0。*Trans* 中的 *arJ* 还未确定。最后将该系统头指针和当前指针指向该 *SysNode*。转入 Step2，初始化 *SyS* 中各项为空。

Step2 *JobArray* 数组中，所有的数字都大于 0，则表示当前的 *SySNode* 为中止状态。若当前状态为中止状态，转入 Step3，否则，转入 Step4。

Step3 当前状态为终止状态后，则 *Trans* 中的 *arT* 设为(0,0,⋯,0)，*value* 为 0，*arJ* 为(*M*,*M*,⋯,*M*)，转入 Step8。

Step4 对于系统中的 *SySNode*，计算 *JobArray* 和 *TimeArray* 数组。首先，从当前状态的 *arJ* 中，确定能够进行操作的工序数，每个工序能够进行的模式的数量，那么下一状态的数量就是所有可以进行操作的工序的模式数的乘积。然后检测 $arJ[i]>0, arJ[i]<M$，则在 *Trans* 中的 *arJ*[*i*]=*M*，该工序在当前时刻处于激活工作状态，则在下一状态中，该工序就已完成。子算法(3.2 节中的(1))用于检测某状态是否满足约束条件。新的状态节点存储在一栈中，并且每一状态都有一个栈。转入 Step5。

Step5 弹出当前栈顶元素，若栈为空，转入 Step7，否则转入 Step6。

Step6 栈顶弹出的状态，计算从当前状态到新状态的转移值。转入 Step8。

Step7 系统状态节点 *SySNode* 的 *Trans* 中的 *arJ* 都暂时存储于栈中，并且整个算法是递归的。当前栈为空时，返回到当前的 *SySNode* 所在的栈中。转入 Step10。

Step8 检测新状态是否在系统 sys 中，若是，转入 Step11，否则转入 Step9。

Step9 计算当前状态到新状态的数值 *value*。由于新的状态已经在系统 *SyS* 中，*value* 值已知，并且状态间的转移值已知，因此采用 *SySNode* 转移算法来计算当前状态的 *value*。转入 Step5。

Step10 检测包含 *SySNode* 的栈是否存在，若存在，转入 Step5，否则转入 Step12。

Step11 产生新的 *SySNode*，*arJ* 即为新的状态，剩余时间由子算法(3.2 节中的(3))来确定。新的 *SySNode* 插入系统 *SyS* 中，并将当前指针指向 *SySNode*。转入 Step2。

Step12 算法终结。信息存储于 *SyS* 中。

3.2 子算法

(1)约束条件检测

输入为系统 *SyS* 链表，新生成的状态 *state* 和所有工序的信息。在新状态中，所有工序所需的资源数要小于系统的资源数。

$$\sum\gamma_i r_{i,j}(m)\le R_j$$

其中，i 是工序号；m 是工序中模式；j 是资源序列；γ 是 0-1 函数，工序为处理加工状态时，设为 1，否则为 0。每一工序都有一个不兼容序列，对活动集，对照工序中的不兼容集，即可判断是否满足该条件。最后检测活动是否有效，即新的状态相对于现有状态，至少有一个工序加工发生。

(2)剩余时间算法

在该算法中，输入为现有系统 *SyS*，新状态 *state*，输出为新状态的剩余时间序列。初始化剩余时间序列与当前系统状态 *Trans* 中的 *TimeArray* 相同。然后再对新序列按下列准则

进行更新：在新状态中，找出满足 $arJ[i]>0$，$arJ[i]<M$ 的 i 集合，再在集合中找出使 arT 最小的值 t，最终的剩余时间即为时间序列中 i 集合的时间减去 t，其余时间不变。

(3)状态转移时间算法

输入为两个 *SySNode*(节点 *sysnode*1 和 *sysnode*2)，在 *sysnode*1 中一个 *Trans* 的 *arJ* 和 *SySNode* 中的 *arJ* 相同。*sysnode*2 的所有信息都已获得，包括 *Trans* 中的 *value* 值，则算法为计算 *sysnode*1 中 *Trans* 的 *value* 值。

从工序信息里能够获得加工时的时间随机分布，转移时间依靠该分布概率来获得。设 *sysnode*1 的状态为 *arJ*1，*sysnode*2 的状态为 *arJ*2。首先根据 *arJ*1 和 *arJ*2 的变化计算所有可能的组合。例如：假设系统有 2 个新工序 *job*1,*job*2，*job*1 位于模式 1(0,0.4,0.3,0.3)，*job*2 位于模式(0,1)，则系统有 8 种组合。排除掉 0 组合，则有用的组合为(0.4,0.3,0.3)。总的传递时间是所有可能结果的和。对每一概率，局部传递时间都能从剩余时间来获得。

4 应用程序执行

采用上述算法，利用 MicroSoft Visiusl Studio C++ 7.0 设计了下述应用程序。系统输入的工序集如表 1 所示。并且系统总的资源数为(7,8)。从表 1 可知，对于 0 号工序，没有前置工序，不兼容工序为 1，有 2 种加工模式 0 和 1，需要的资源数分别为(2,3)和(1,2)，时间分布概率为(0.1,0.5,0.4)和(0,0,0.4,0.5,0.1)。将表 1 中数据输入程序并运行后得到图 1 的结果。从图 1 可知，平均加工时间最优的序列为[0 0 0 0], [1 0 0 0], [99 1 1 0], [99 99 99 2]，时间值为 3.7。

表 1 工序输入数据

工序.	前置工序	不兼容工序	模式	资源	加工时间分布
0	…	1	0	(2,3)	0.1,0.5,0.4
			1	(1,2)	0,0.4,0.5,0.1
1	…	0	0	(2,4)	0,0.8,0.2
2	0	3	0	(3,3)	0,0.5,0.5
3	1	2	0	(2,5)	0,0.3,0.7
			1	(3,6)	0.2,0.8

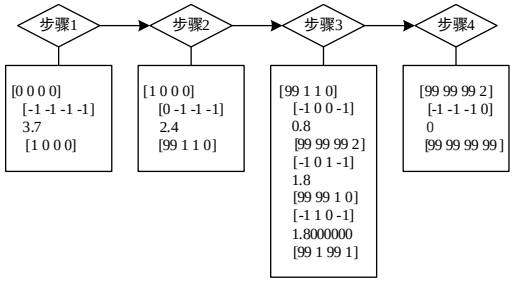

图 1 结果流程

5 结束语

本文考虑了不兼容的约束和多种加工模式，其中每种模式与一系列确定的资源需求和离散的加工时间分布相关联。提出了一种用离散随机动态系统描述的基于加工时间离散随机分布的 RCPSP 模型，并采用一个动态规划算法进行求解，使得在满足资源约束和工序约束的前提下，总的平均加工时间最短。

参考文献

[1] Brucker P, Drexel A, Mohring R H. Resource-constraint Project Scheduling: Notation, Classification, Models, and Methods[J]. Eur. J. Oper. Res., 1999, 112(1): 3-41.

[2] Yang B, Genunes J, O'Brien W J. Resouce Constrained Project Scheduling: Past Work and New Directions[D]. Florida, USA: University of Florida, 2001: 2001-2006.

[3] Neumann K, Schwindt C, Zimmermann J. Resource-constrained Project Scheduling with Time Windows: Prespectives in Modern Project Scheduling[M]. New York, USA: Springer, 2006: 375-407.

[4] 程　序, 吴　澄. 一种复杂项目调度问题的混合智能算法[J]. 计算机集成制造系统, 2006, 12(4): 585-589.

[5] 胡奇英, 刘建庸. 马尔科夫决策过程引论[M]. 西安: 西安电子科技大学出版社, 2000.

(上接第 12 页)

从图中不难发现，有的状态转移路径多，出现饱和的机率就大，有的状态出现的概率则非常小，如果采用自然路径采集算法，为使状态S_8达到饱和，会出现其他状态饱和后出现大量的空转以帮助状态S_8达到饱和。在某些特殊设计情况下，甚至令自然路径采集算法驱动所有状态达到饱和的时间无法接受。

6 结束语

虽然“克隆”法逆向分析出的芯片和原芯片完全一致，但对不少逻辑器件却无能为力，另外即使“克隆”出了用于替换电路系统中原位置的器件，进一步可能具有原电路系统完全复制的能力，但这将引起产权纠纷，且对电子设备的自动诊断与维修没有任何直接意义，对开展被诊断维修设备的 TPS 开发无任何帮助。

逻辑分析法则是通过输入不同组合不同序列的向量激励信号，在输出端收集其对应的变化输出，对收集的数据进行逻辑综合处理最终推导出引脚间逻辑功能。该方法实质上穷举模拟了该逻辑芯片在电路系统中全部可能的工作状态，逆向分析过程对 PLD 器件无任何损害，分析出的引脚属性和引脚间逻辑功能有助于 TPS 的开发和维修工程师完成电子设备的自动诊断与维护。

大量实践证明，本文所提出的分状态路径驱动数据采集算法，能够快速有效地解决逆向分析同步时序型 PLD 中所面临的时间和空间问题，克服了自然路径采集算法驱动所有状态达到饱和时在时间上可能出现无法接受的问题。本文所讨论的算法已在笔者设计实现的 PLD 逆向分析系统中成功应用，对于提高系统的性能和解析的成功率发挥了重要作用。

参考文献

[1] 边计年, 薛宏熙. 数字系统设计自动化[M]. 北京: 清华大学出版社, 2005.

[2] 简弘伦. 精通 Verilog HDL: IC 设计核心技术实例详解[M]. 北京: 电子工业出版社, 2005-10.

[3] 张　平, 李清宝. 加密可编程逻辑阵列芯片引脚的判别[J]. 电子技术, 2002, 29(1): 3-4.

[4] 周保华. 可程式逻辑元件逆向工程之研究[J]. 大叶学报, 2001, 10(1): 79-86.

[5] 王　炜, 李清宝. 一种快速逻辑综合算法[J]. 计算机工程与应用, 2000, 36(6): 84-86.

[6] 翟献军, 肖梓祥. 一种改进的快速逻辑综合算法[J]. 计算机应用研究, 2002, 19(1): 40-41.