文章编号：1000－8152(2005)02－0229－08

考虑缓冲区的自动生产单元的无死锁调度策略

徐 刚，吴智铭
（上海交通大学 自动化系，上海 200030）

摘要：在制造系统中，必须防止死锁的发生．本文提出了一种在制造系统(带有有限缓冲区)中搜索最优的无死锁调度的算法．为此首先介绍了死锁问题及其图论表示方法，然后在遗传算法的基础上，运用图论算法来保证无死锁的调度结果．为了保证遗传算法生成的调度策略能够满足所要求的约束，运用图论方法选择无死锁个体，或添加缓冲区，从而在基本保证了系统的主要性能指标的同时，得到系统可行的无死锁调度结果．最后给出了一个运用此方法解决死锁问题的实例．

关键词：遗传算法；图论；无死锁调度
中图分类号：TP278 **文献标识码**：A

Deadlock-free scheduling method for automated production cell including buffer

XU Gang, WU Zhi-ming
(Department of Automation, Shanghai Jiao Tong University, Shanghai 200030, China)

Abstract: Deadlock must be avoided in a manufacturing system. In this paper, an efficient algorithm for finding an optimal deadlock-free schedules in a Manufacturing System with very limited buffer is presented. First, the deadlock problem and its graph theory representation is introduced. Then based on the effective genetic algorithm (GA) search method, the graph theory is introduced to assure deadlock-free. In order to make the scheduling strategy generated by GA meet the required constrains, a deadlock detection procedure based on graph theory is involved as a condition to select deadlock－free scheduling, or to allocate the buffer. So the feasible scheduling result is obtained while the main performance indicator is guaranteed. Finally, a case is given to demonstrate the effectiveness of this method.

Key words: genetic algorithm(GA); graph theory; deadlock-free scheduling

1 前言(Introduction)

当根据制造和生产计划对柔性制造系统(FMS)进行有效的设计并使其正常运作时，会给制造企业以极大的帮助．不适当的资源分配会导致死锁．死锁是一种状态，一个零件集合陷入了循环等待，即集合中的每个零件都在等待被同一集合中另外一个零件占据的资源．死锁会引起不必要的损失，如很长的系统等待恢复时间，降低资源的利用率．因此在防治死锁发生的同时，开发一些能有效地改进和优化系统性能的算法是非常重要的．Banaszak[1]，Barkaoui[2]，Abdallah[3]，Ezpeleta[4]，Fanti[5,6]，Wysk[7,8]，Xiong[9]，Wu[10]，Viswanadham[11]提出了基于Petri网的死锁处理策略．遗传算法(GA)是一种有效的全局搜索算法[12～14]．将遗传算法所作的有效的搜索同图论的方法结合起来，在GA的搜索过程中添加更多的限制，从而得到系统的无死锁调度，使问题在执行的进程中得到优化．在本文中，假设自动导向小车(AGV)的数量是无限的．零件进入车间，根据工序安排，在相应的机床上加工，最终加工完毕离开车间．本文的方法相对于已有的调度算法来说，有如下优点：1)可以找到0-缓冲区情况下的无死锁调度；2)可以找到给定数量缓冲区下的无死锁调度；3)可以找到任意数量缓冲区情况下的无死锁调度．可以为工程技术人员提供多种多样的选择，增加了调度实施的灵活性．

2 运用有向图建模(Modelling with directed graph)

本文中的算法是建立在两个有向图的基础上：工作过程有向图 $G_W = (V_w, C_W UD)$ 和变迁有向图 $G_T = (V_T, C_T)$．在图 $G_W = (V_W, C_W UD)$ 中，V_W 代表

收稿日期：2002－10－22；收修改稿日期：2004－03－29．
基金项目：国家自然科学基金资助项目(60074011，70071017)．

一组零件的各工序节点，包括一个源节点 S 和一个终止节点 E，分别代表调度的开始和结束；C_W 表示由工艺路线所规定的工件的各道工序的顺序关系的有向弧的集合，它反映了同一工件的工序之间的次序约束. D 表示共享同一机器的工序之间的互斥关系的无向弧集合. 调度问题就是对图 $G_W = (V_W, C_W \cup D)$ 的操作，为 D 中各个无向弧选择一个相应的方向，得到一个确定性的调度，从而使一个性能指标为最优. $G_W = (V_W, C_W \cup D)$ 的一个例子见图 1(b). 在图 $G_T = (V_T, C_T)$ 中，V_T 代表一组资源(机床，缓冲器) 的节点，C_T 表示由工艺路线所规定的工件的各道工序的资源请求关系的有向弧的集合. 这个图描述的是资源正被一个加工任务占据，并且此加工任务为了下一道工序的进行而请求其他的资源. 所以这个图必须随着加工任务的推进而不断更新. 图 $G_T = (V_T, C_T)$ 的例子如图 1(c)所示. 图 1(a)中用到的变量如下所示：

(i,j)：第 i 个加工任务的第 j 道工序；

$b_{i,j}$：工序(i,j) 的开始时间；

$t_{i,j}$：工序(i,j) 的处理时间；

$m_{i,j}$：工序(i,j) 的机器；

$u_{i,j}(i,j)$：工序(i,j) 在处理该工序的机器上的前序工序. If$(u_{i,j} = S)$，则工序(i,j) 为该机器上的头道工序；

$v_{i,j}(i,j)$：工序(i,j) 在处理该工序的机器上的后序工序. If $(v_{i,j} = E)$，则工序(i,j) 为该机器上的末道工序；

POR：工件 i 的路径中的前序工序，if POR$[(i,j)]$ = NULL，then 工序 (i,j) 是工件 i 的头道工序；

SOR：工件 i 的路径中的后续工序，if SOR$[(i,j)]$ = NULL，then 工序 (i,j) 是工件 i 的末道工序.

以虚线连接的节点序列指示了一台机器上的处理序列，以实线连接的节点序列指示了一个零件的加工工序信息. 得到的邻接矩阵记为 A. 如图 1(a)所示. 图 1(b)所示的状态是有向图的资源互斥关系确定之后，无向的资源互斥关系变为确定性的有向弧，一个调度由此确定下来. 进一步表示成如图 1(c)中所示，M_1 被工件 1 占据，M_2 被工件 2 占据. 工件1申请 M_2，工件2申请 M_1，从而形成了循环等待. 整个系统无法向前推进，图 1(c)对应状态的甘特图如图 1(d)所示. 其中，虚线表示的是不可到达的位置，因为在此调度序列中出现了死锁. 图 1(e)为调整调度序列后得到的甘特图. 由此图可以看出，调度序列中不会出现循环等待的情况，也即此调度序列是无死锁的. 因为图 1(e)中是加工任务的串行排列(即加工任务 1 所有工序完成后，进行加工任务 2 的加工)，所以此调度序列的流程时间(makespan)很大. 如果将某个零件放入缓冲器中，释放其占据的资源，则系统可以重新运行. 如图 1(f)所示，加工任务 1 在完成工序 1 之后，进入缓冲区，从而解除了循环等待的情况，并且调度结果的流程时间与图 1(b)中是相同的. 缓冲器的作用就是临时存放零件，让出占据的资源. 本文中的方法，主要是在保证调度序列无死锁的情况下，尽量减小调度的流程时间.

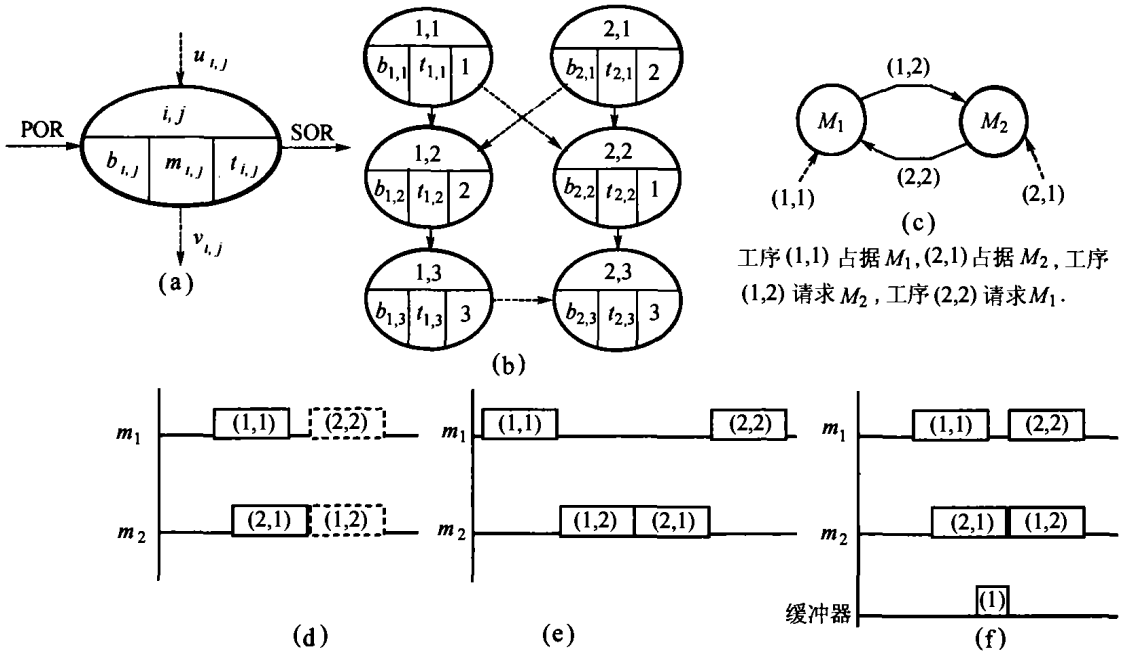

图 1 图论表示形式

Fig. 1 Representation using graph theory

3　算法的描述(Description of the algorithm)

算法从系统初始有向图模型(析取弧的方向未确定)中得到加工任务的初始信息,作为染色体编码的依据.编码后,进行遗传操作.到达结束条件后,染色体经过解码,将析取弧的方向确定下来,形成一个调度方案,并修正相应图的邻接矩阵 A.给定一个全局时钟,通过对矩阵 A 的分析,得到对应变迁有向图的资源请求矩阵 M.然后确定图中是否有死锁,并根据基于矩阵 M 的检测算法判定调度是否可行.若 M 中无死锁,算法继续进行,若有死锁,则此调度不可行.形成死锁的某个加工任务将会进入缓冲,若此时死锁消除,则继续,否则再选择其他的加工任务.这样循环等待的条件就被打破了.初始的调度结果就得到了修正.最终得到无死锁的调度.

本文有几种方法来避免调度中死锁的发生:

1) 删除出现死锁的调度,从种群中选择一个新的调度;

2) 添加缓冲区来暂时地解决冲突;

3) 调整零件的投放时间和驻留时间(在缓冲区中的停留时间)以避免资源竞争和死锁.

这 3 种方法作为子模块配置在 GA 调度程序中,根据系统需要调用.

1) 从图中得到初始的邻接矩阵 A.

A 是对应图 $G_W = (V_W, C_W \mathrm{U} D)$ 中节点的结构体数组,结构体中的成员已列在了第 2 部分.在这些结构体的成员中,(i,j), $t_{i,j}$, $m_{i,j}$, POR, SOR 是由初始的加工任务确定的. $b_{i,j}, u_{i,j}, v_{i,j}$ 是当一个调度形成之后确定下来的.

2) 染色体的编码.

在矩阵 A 中搜索工序(i,j),此工序对应机床 $m_{i,j}$.找到的(i,j) 的集合形成 C.C 中包含了染色体的基本信息.采用一系列的并行排列的机床来代表染色体,每台机床是染色体中的一个矢量,它包含了这台机床的工序安排.机床上的每道工序用三项来表示:任务号,该任务的工序号,该工序的开工时间.开工时间的计算考虑了资源的约束.

染色体的编码方式如下:

$$\{C_{chm}\}_{chm=1}^{NG}.$$

其中

$$C_{chm} = \{\{P_{mq_m}\}_{q_m=1}^{Qm}, \{PO_{mq_m}\}_{q_m=1}^{Qm}, \{ST_{mq_m}\}_{q_m=1}^{Qm}\}_{m=1}^{M}.$$

C_{chm} 为染色体,NG 为每一代中染色体的数量,Q_m 为每台机床上的工序数量,M 为机床的数量,P_{mq_m} 为零件号,PO_{mq_m} 为该任务的工序号,ST_{mq_m} 为该工序的开工时间.

3) 遗传操作.

A) 交叉:

i) 从种群中选定参加交叉的两个个体 parent1, parent2(最优个体),随机选定一个机床号 m;

ii) 构建 child1(child2),拷贝 parent1 (parent2)上 m 号机床的工序到 child1(child2)相同的机床上;

iii) 顺序完成其他的机床,parent1(parent2)的相应机床工序换到 child2(child1)上相应的机床工序.

B) 变异:

染色体中同一台机器内处理工序之间的随机互换.

4) 开工时间.

染色体的第一代个体由随机产生,每个工序的开工时间计算如下:

st_{MQ}: 当前工序的开工时间;

stj_{MQ}: 同一加工任务前道工序(POR)的开工时间;

ptj_{MQ}: 同一加工任务前道工序(POR)的工序时间;

stm_{MQ}:同一机床上前道工序 $u_{i,j}(i,j)$ 的开工时间;

ptm_{MQ}: 同一机床上前道工序 $u_{i,j}(i,j)$ 的工序时间;

· 若此工序是加工任务的第一道工序,并且是机床上的第一道工序,则 $st_{MQ} = 0$;

· 若此工序是加工任务的第一道工序,但不是机床上的第一道工序,则 $st_{MQ} = stm_{MQ} + ptm_{MQ}$;

· 若此工序不是加工任务的第一道工序,是机床上的第一道工序, $st_{MQ} = stj_{MQ} + ptj_{MQ}$;

· 若此工序不是加工任务的第一道工序,并且也不是机床上的第一道工序, $st_{MQ} = \max(stm_{MQ} + ptm_{MQ}, stj_{MQ} + ptj_{MQ})$.

5) 染色体的解码.

利用上面得到的染色体 $C_{chm} = \{\{P_{mq_m}\}_{q_m=1}^{Qm}, \{PO_{mq_m}\}_{q_m=1}^{Qm}, \{ST_{mq_m}\}_{q_m=1}^{Qm}\}_{m=1}^{M}$ 及工序的开工时间,修正矩阵 A,即形成一个确定性的调度,矩阵 A 中的 $u_{i,j}, v_{i,j}$ 将会确定下来.更确切的说,在图 $G_W = (V_W, C_W \mathrm{U} D)$ 中集合 D 中的弧的方向将会被确定下来.

6) 图中的操作.

要进行死锁检测,必须得到变迁有向图.变迁有向图由工作过程有向图得到.在 makespan 中一个给定的时间 T,选择已经开始并且还没有结束的工序

(i,j),或者是刚刚开始加工的工序(i,j),以及在同一个加工路线上的他们的下一道工序.同时可以确定工序(i,j)占据的机床m_i,工序(i,j)在同一工艺路线上的下道工序请求的资源m_j,m_im_j之间的请求关系将会添加到符号矩阵M.运用文献[8,15]中的方法对矩阵A进行死锁检测操作.

为了防止死锁发生,某些加工任务将会被送入缓冲区.进入缓冲区的工件应是属于DL集合.并且是最先完成当前工序的工件.进入缓冲区的时间是此工件完成当前工序的时间,出缓冲区的时间为此工件的下一道工序开工的时间.由此,可以将缓冲区的调配考虑到原始调度方案中,在保持原始调度方案的makespan基本不变的条件下,避免了原始调度中的死锁.假定在整个任务调度过程中,为避免系统中的死锁和冲突以及要获得一个小的makespan值,分配缓冲区的数量上限可以达到Bm,Bm是执行生产过程的一个重要指标.

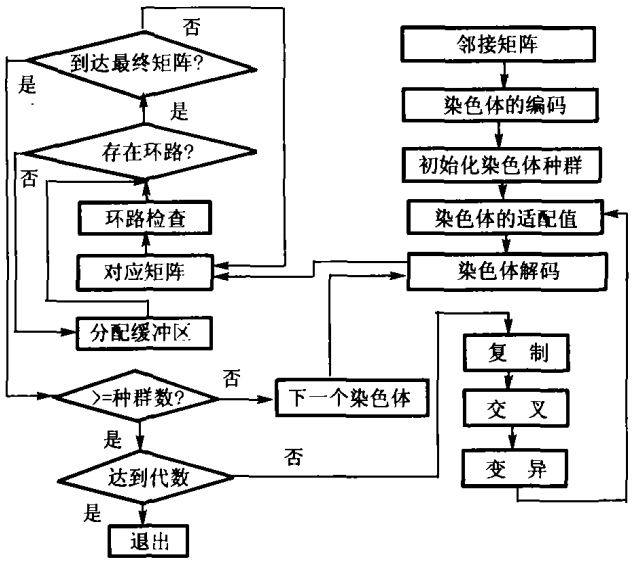

图2 算法框图

Fig. 2 Algorithm frame

如果车间中缓冲区的数量少于理想调度下的Bm,在嵌入的任务调度运行到某个步骤时,已完成某道工序的机床不能够进入一个空的缓冲区释放占据的机床资源.在这种有限缓冲区的情况下,车间中会出现一个临时的阻塞和死锁.要处理如下情况,需要采用如下措施:

1) 停止从装卸台输入新任务,以减少车间中的空间占用;

2) 等待其他任务完成处理,释放一个缓冲来解决阻塞状态.只要阻塞一解除,新的任务就可以重新输入.

如果占用机床的任务都完成了处理,没有缓冲区释放,可以判定这个测试的调度就是死锁的,应被删除.选择一个新的染色体重新试验.算法中考虑了缓冲区的循环利用问题.

具体算法简略描述如下:

a) 给定一个全局时钟T;

b) If $0 \leqslant T - b_{i,j} \leqslant t_{i,j}$(对$A$中的每一个工序) Then{

If 矩阵M中只有一个元素或者矩阵M中的元素不发生变化

Then 跳到loop:

Else 修正矩阵M并且运用文献[18]中的方法对矩阵M进行回路检测.

}

If 矩阵M中有死锁,导致死锁的加工任务的工序被放入集合DL.将DL中的工序分为前向工序集FC和后向工序集BC.BC中的工序对应FC中的工序.

Then{将工序(i,j)记为v_h, Flag-inBuffer记为零件进入缓冲的标志, $T_{L_{ST}}$[BufferNum][Part-Num1]记为加工任务离开缓冲区的时间,Buffer-Num记为缓冲区的编号.

加工任务v_h的编号是JobNum1;

FOR 系统中的每个缓冲区 Buffer-Num

{ If $T - b_{i,j} = 0$ 和 Flag-inBuffer[Buffer-Num][JobNum1] = 1

{Flag-inBuffer[BufferNum][JobNum1] = 0;

$T_{L_{ST}}$[BufferNum][JobNum1] = b_{ij}; //加工任务JobNum1离开缓冲区;

修正矩阵A.

}

}

① 在矩阵A中检索对应DL中的v_h,得到子集$\{v_{h_1}, v_{h_2}, \cdots\}$,它的工序开工时间是$\{b_{v_{h_1}}, b_{v_{h_2}}, \cdots\}$,并得到工序时间$\{t_{ph_1}, t_{ph_2}, \cdots\}$.对$FC$中的工序,计算$\min\{b_{v_{h_1}} + t_{ph_1}, b_{v_{h_2}} + t_{ph_2}, \cdots\}(v_{h_i} \in FC)$,对应$\min\{b_{v_{h_1}} + t_{ph_1}, b_{v_{h_2}} + t_{ph_2} \cdots\}$的加工任务将会进入缓冲器.如果$\min\{b_{v_{h_1}} + t_{ph_1}, b_{v_{h_2}} + t_{ph_2}, \cdots\}$有相同的值,则计算$\min\{b_{v_{h_1}}, b_{v_{h_2}}, \cdots\}(v_{h_i} \in BC)$,对应$\min\{b_{v_{h_1}}, b_{v_{h_2}}, \cdots\}(v_{h_i} \in BC)$的加工任务将会进入缓冲区.如果在$\min\{b_{v_{h_1}} + t_{ph_1}, b_{v_{h_2}} + t_{ph_2}, \cdots\}(v_{h_i} \in FC)$和$\min\{b_{v_{h_1}}, b_{v_{h_2}}, \cdots\}(v_{h_i} \in BC)$都有相同的值,

进入缓冲区的加工任务可以随机选择. 如果对 BC, FC 中的每个工序: $b_{v_{FC}} + t_{p_{FC}} = b_{v_{BC}}$, 则 Time-All-Delay = RandomTime, 开工时间将会增加 Time-All-Delay. 否则 Time-All-Delay = 0.

② FOR 每个缓冲区 B-Order

若有缓冲区为空, 进入相应的缓冲区, BufferNum = BufferNum - 1;

若所有缓冲区都被占据, 则重新创建新的缓冲区, BufferNum = BufferNum + 1;

(对缓冲区数目 Bm 给定的情况, 若所有缓冲区都被占据, 即 BufferNum = Bm 时, 则舍弃此调度方案, 重新选择调度方案.)

③ 设置零件进入缓冲区的标志: Flag-inBuffer [BufferNum][JobNum] = 1.

④ 为开工工序的所有开工时间增加 Time-All-Delay;

⑤ $T_{B_{ST}}$[BufferNum][PartNum1] = $b_v + t_p$; //零件进入缓冲区的时间

⑥ 修正矩阵 M, A;

}

}

loop: T ++

Go to b) while (T < makespan).

}

· 运用上面的算法, 可以得到最优调度需要的缓冲区的数目及包含缓冲区的任务调度甘特图.

· 给定缓冲区的数目 Bm, 运用上面的算法可以得到对应此 Bm 的最优调度, 以及包含缓冲区的任务调度甘特图.

从以上算法描述可以看出, 解除死锁的措施并没有改变调度的流程时间(makespan), 系统的性能指标在此表现为 makespan. 添加缓冲区的目的主要是为了改善系统的行为特性: 死锁, 即在已有的调度序列的基础上通过添加缓冲区解决调度序列中的死锁, 或是直接选择无死锁的调度序列. 使得到的调度结果在投入车间后可以切实地得到运行.

7) 适应值函数.

使 makespan 最小, 每个染色体有 M 台机床, 计算每台机床完成全部工序的时间, 从 M 台机床中选择完成时间的最大值. 在机床上选取最后一道工序的开工时间, 加上此道工序的处理时间. 对此染色体中的所有机床重复上述两个步骤. 则此染色体的 makespan 计算如下:

$$\max[(ST_{MQm} + PT_{MQm})_1, (ST_{MQm} + PT_{MQm})_2, \cdots (ST_{MQm} + PT_{MQm})_M].$$

4 实例分析(Case study)

任务需求说明见表 1、图 3(机床号后面对应的为工序需要的时间, 单位为 s).

表 1　调度任务

Table 1　Scheduling jobs

J_1	J_2	J_3	J_4
M_1:40	M_2:45	M_1:212	M_3:55
M_2:100	M_1:65	M_2:73	M_2:65
M_3:36	M_3:98	M_3:32	M_1:35

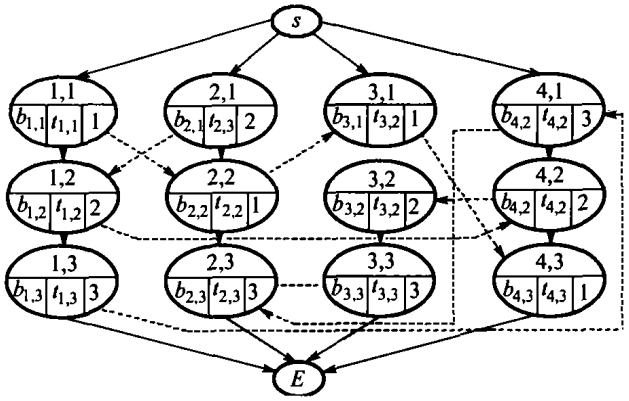

图 3　有向图的调度表示形式

Fig. 3　Scheduling representation using directed graph

t = 0 s 时, 工序(1,1) 和(2,1) 同时开工, 利用资源请求图 M (对应图 4 中的图)可得到下面的死锁检测算法:

$$\begin{pmatrix} & m_1 & m_2 & m_3 \\ m_1 & 0 & 12 & 0 \\ m_2 & 21 & 0 & 0 \\ m_3 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} & m_1 & m_2 & m_3 \\ m_1 & 0 & 12 & 0 \\ m_2 & 21 & 0 & 0 \\ m_3 & 0 & 0 & 0 \end{pmatrix} \rightarrow$$

$$\begin{pmatrix} 121 & 0 & - \\ 0 & 0 & - \\ - & - & - \end{pmatrix}.$$

从上面的计算可知 Job1, Job2 在机床 $M1$, $M2$ 处存在循环等待关系. 加工任务 Job1, Job2 完成当前工序后, 陷入循环等待. 若加工任务 1, 2 都不释放资源, 则系统就会陷入局部死锁. 不加以解决, 系统就会陷入全局死锁(若加工任务 4 的工序一完成后). 因为 $b_{(1,1)} + t_{(1,1)} < b_{(2,1)} + t_{(2,1)}$, 则在 t = 40 s 时, 将 Job1 放入缓冲区, 从而机床 M_1 空闲, Job2 可以获得机床 M_1 进入下一步工序加工, 死锁的情形得以解除. 在 t = 45 s 时, 系统状态如图 4,5 所示.

取交叉概率为 prob-of-crssover = 0.7, 变异概率为 prob-of-mutate = 0.2, 最大个体数量为 num-of-individual = 10,

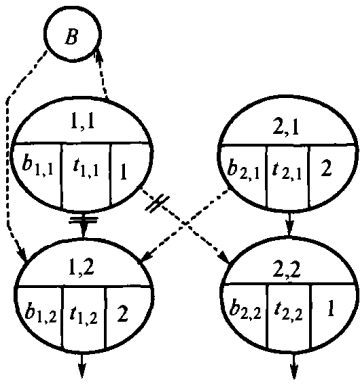

图 4 Job1 进入缓冲区

Fig. 4 Job1 enter into buffer

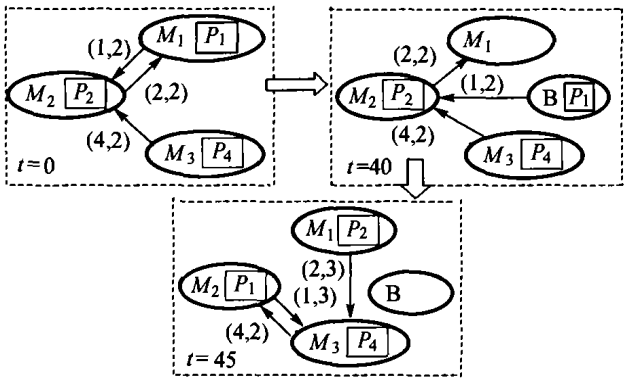

图 5 缓冲区解决死锁

Fig. 5 Resolute deadlock with buffer

经过 100 代后，算法 A 收敛于 427. 采用 C 编码，奔 III - Based PC(566MHz)，64M RAM. 算法运行时间为 5 s. 运用此算法，得到的最小 makespan 的调度为 427. 它的 Gannt 图如图 6 所示. 此调度需要一个缓冲区. 从 100 代个体中选择前 5 个个体(为使 makespan 最小)，其调度结果如表 2 所示.

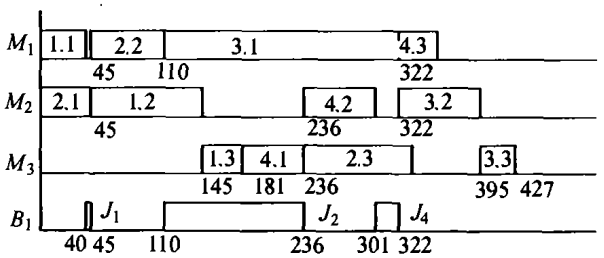

图 6 Makespan = 427 的调度

Fig. 6 Scheduling with makespan = 427

表 2 调度结果

Table 2 Scheduling result

makespan	427	449	469	501	512
buffer-num	1	1	1	1	0
buffer-time	152	98	85	46	0

buffer-time 为工件在缓冲区中的总时间；buffer-time 为 0 表示此调度方案为无死锁调度；buffer-num = 0 表示此调度方案不需要缓冲区. 用户可以根据自己的需要，从上面的调度结果中进行选择. Makespan 最小的无死锁调度方案如图 7 所示. 将加工任务增大，如表 3 所示. 得到的结果如图 8 ~ 11 所示.

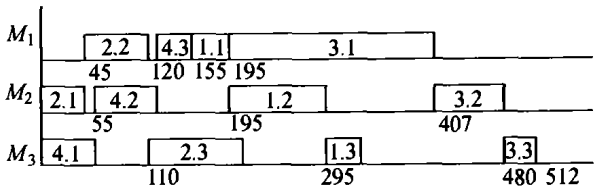

图 7 Makespan 为 512 的无死锁调度

Fig. 7 Deadlock free scheduling with makespan = 512

表 3 加工任务信息

Table 3 Jobs information

J_1	J_2	J_3	J_4	J_5	J_6
M_1:40	M_2:45	M_1:212	M_3:55	M_1:50	M_2:95
M_2:100	M_1:65	M_2:73	M_2:65	M_3:120	M_1:50
M_3:36	M_3:98	M_3:32	M_1:35	M_2:30	M_3:40

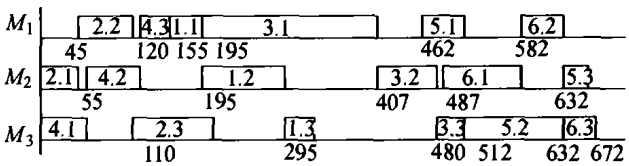

图 8 0-Buffer 情况的 deadlock-free 调度结果. makespan = 672

Fig. 8 Deadlock-free scheduling with 0-Buffer. makespan = 672

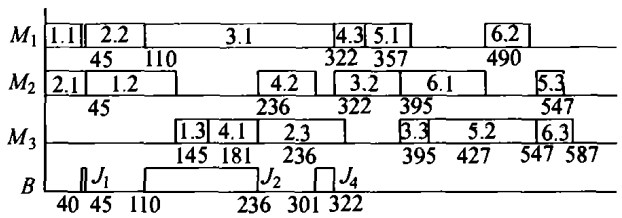

图 9 1-Buffer 情况的 deadlock-free 调度结果. makespan = 587

Fig. 9 Deadlock-free scheduling with 1-Buffer. makespan = 587

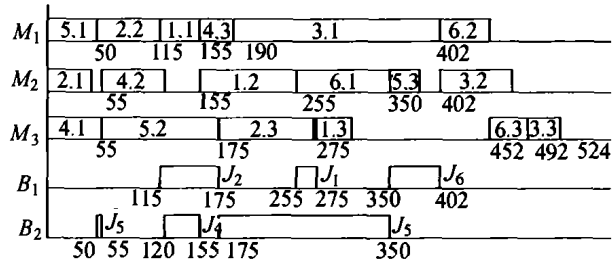

图 10 2-Buffer 情况的 deadlock-free 调度结果 (makespan = 524)

Fig. 10 Deadlock-free scheduling with 2-Buffer (makespan = 524)

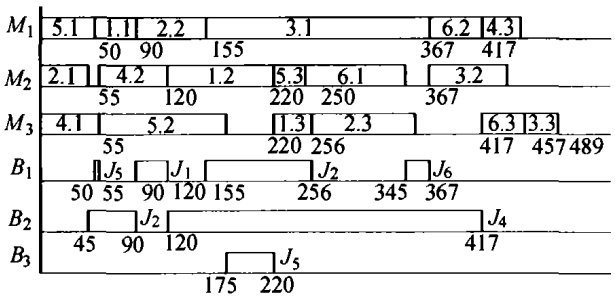

图 11　3-Buffer 情况的 deadlock-free 调度结果 (makespan = 489)

Fig. 11　Deadlock-free scheduling with 3-Buffer (makespan = 489)

从以上结果可以看出,随着缓冲区数量的增大,无死锁调度允许的 makespan 值逐渐减小,效果比较明显.可以看出,给定加工任务,适当增加系统缓冲区数量,可以减少批工件的 makespan 时间.

运用 Fisher and Thompson[15] 的实例来进行测试:10 个任务 10 台机床(10 × 10 问题)以及 20 个任务 5 台机床 (20 × 5 问题).对 10 × 10 问题,算法运行 10 次,有 7 次得到最优的 makespan 值:930,需要 4 个缓冲区,平均的运行时间为 6.75 s,makespan 的平均值为 941.8;对 20 × 5 问题,算法运行 10 次,有 6 次得到最优值:1165,需要 4 个缓冲区,平均的运行时间为 10.64 s,得到的 makespan 的平均值为 1189.7.本文中的方法主要侧重于解决调度中的死锁问题,找到无死锁的调度序列,或是将缓冲区考虑进来,解决调度序列中的死锁问题.与 Chen[16] 以及 Tsujimura[17] 要解决的问题不同,本文的方法是在得到调度结果的基础上,又前进了一步.在遗传算法中嵌入了一个检查和修补过程.由 Chen 及 Tsujimura 方法得到的只是一个简单的调度序列,此调度序列中有没有死锁,不得而知.但是依靠本方法,可以对调度序列进行检查,如果有死锁则可以对调度进行修补,即添加缓冲区来处理死锁.

表 4　Muth and Thompson 的实例

Table 4　Case of Muth and Thompson

	10 × 10 问题	20 × 5 问题
Chen Xiong(2002)	930	1173
Tsujimura (2000)	930	1178
本方法(deadlock-free)	930(4)	1165(4)

表 5　10 × 10 问题解的分布

Table 5　Distribution of solution for 10 × 10

流程时间	930 – 940	940 – 950	950 – 960	960 – 970	970 – 980	980 – 990	990 – 1000	1000 – 1010	1010 – 1020	1020 – 1030	1030 – 1040
解的数量	12	85	137	204	143	154	135	97	67	45	34
需要的缓冲区的数量的平均值	3.4	3.3	2.7	2.1	1.9	1.3	1.2	1.2	1.1	1.0	0.7

表 6　20 × 5 问题解的分布

Table 6　Distribution of solution for 20 × 5

流程时间	1160 – 1170	1170 – 1180	1180 – 1190	1190 – 1200	1200 – 1210	1210 – 1220	1220 – 1230	1230 – 1240	1240 – 1250	1250 – 1260	1260 – 1270
解的数量	9	23	172	257	117	215	187	89	76	58	56
区域划分	1	2	3	4	5	6	7	8	9	10	11
需要的缓冲区的数量的平均值	3.7	3.1	2.6	2.4	2.1	2.0	1.6	1.4	1.1	1.1	0.6

由表可以看出,流程时间越大,需要的缓冲区的数量越小.即带有零缓冲区的无死锁调度随着流程时间的增大出现的机会越多.对 20 × 5 问题,采用如表 7 所示的参数,与表 4 中算例采用相同的系统配置,得到最优解的最少时间为 9.4 s:

表 7　图 19 中 GA 的参数

Table 7　Parameters of GA for Fig. 19

交叉概率	变异概率	种群数目
0.7	0.2	80

5　结论(Conclusion)

本文基于系统的有向图模型,运用遗传算法有效的搜索的特性,提出了一种考虑系统缓冲区的调度方法;为了保证遗传算法生成的调度能够满足所要求的约束,运用图论方法选择无死锁个体,在必要时添加缓冲区,从而在保证了系统的主要性能指标的同时,得到系统可行的调度结果.运用本文的方法可以得到多种多样的解,这就为工程技术人员提供了多种多样的选择.

参考文献(References):

[1] BANASZAK Z A, KROGH B H. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows [J]. *IEEE Trans on Robotics and Automation*, 1990, 6(6): 724 – 734.

[2] BARKAOUI K, ABDALLAH B I. A deadlock prevention method for a class of FMS [C]// *Proc of IEEE Int Conf on Systems, Man, and Cybernetics*. USA: IEEE Press, 1995: 4119 – 4124.

[3] ABDALLAH B I, EIMARAGHY H. Deadlock prevention and avoidance in FMS: A Petri net-based approach [J]. *Int J of Advanced Manufacturing Technology*, 1998, 16(1): 441 – 451.

[4] EZPELETA J, COLOM J M, MARTINEZ J. A Petri net based deadlock prevention policy for flexible manufacturing systems [J]. *IEEE Trans on Robotics and Automation*, 1995, 11(2): 173 – 184.

[5] FANTI M P, MAIONE B, MASCOLO S, et al. Event-based feedback control for deadlock avoidance in flexible production systems [J]. *IEEE Trans on Robotics and Automation*, 1997, 13(3): 347 – 363.

[6] FANTI M P, MAIONE B, TURCHIANO B. Event control for deadlock avoidance in assembly systems [C]// *Proc of IEEE Int Conf on Systems, Man, and Cybernetics*. USA: IEEE Press, 1997, 4: 3756 – 3761.

[7] HYUENBO C, KUMARAN T K, WYSK R A. Graph-theoretic deadlock detection and resolution for flexible manufacturing systems [J]. *IEEE Trans on Robotics and Automation*, 1995, 11(3): 413 – 421.

[8] WYSK R A, YANG N S, JOSHI S. Detection of deadlocks in flexible manufacturing cells [J]. *IEEE Trans on Robotics and Automation*, 1991, 7(6): 853 – 859.

[9] XIONG H H, ZHOU Mengchu. Deadlock-free scheduling of an automated manufacturing system based on Petri nets [C]// *Proc of IEEE Int Conf on Robotics and Automation*. Canada: IEEE Press, 1997, 2: 945 – 950.

[10] WU Naiqi, ZHOU Mengchu. Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems [J]. *IEEE Trans on Robotics and Automation*, 2001, 17(5): 658 – 669.

[11] VISWANADHAM N, NARAHARI Y, JOHNSON T L. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models [J]. *IEEE Trans on Robotics and Automation*, 1990, 6(6): 713 – 723.

[12] ZHANG F, ZHANG Y F, Nee A Y C. Using genetic algorithms in process planning for job shop machining [J]. *IEEE Trans on Evolutionary Computation*, 1997, 1(4): 278 – 289.

[13] GEN M, TSUJIMURA Y, KUBOTA E. Solving job-shop scheduling problems by genetic algorithm [C]// *Proc of IEEE Int Conf on Systems, Man, and Cybernetics*. USA: IEEE Press, 1994, 2: 1577 – 1582.

[14] SONG Y, HUGHES J G. A genetic algorithm with a machine order-based representation scheme for a class of job shop scheduling problem [C]// *Proc of American Control Conference*. USA: IEEE Press, 1999, 2: 895 – 899.

[15] FISHER H, THOMPSON G L. *Industrial Scheduling* [M]. England: Engle-Wood Cliffs, 1963: 225 – 251.

[16] CHEN Xiong, KONG Qingsheng, WU Qidi. Hybrid algorithm for job-shop scheduling problem [C]// *Proc of the 4th World Congress on Intelligent Control and Automation*. Shanghai: IEEE Press, 2003, 3: 1739 – 1743.

[17] TSUJIMURA Y, MAFUNE Y, GEN M. Introducing co-evolution and sub-evolution processes into genetic algorithm for job-shop scheduling [C]// *Proc of the 26th Annual Conf on the IEEE Industrial Electronics Society*. Japan: IEEE Press, 2000: 2827 – 2830.

作者简介:

徐 刚 (1974—),男,上海交通大学博士研究生,主要从事柔性制造系统中的调度和控制的研究,E-mail: xu.gang@zte.com.cn;

吴智铭 上海交通大学教授,研究方向为离散事件动态系统.