文章编号:1006-5911(2004)07-0820-05

用双向收敛蚁群算法解作业车间调度问题

王常青,操云甫,戴国忠
(中国科学院 软件所智能工程实验室,北京 100080)

摘 要:为了合理高效地调度资源,解决组合优化问题,在Job-Shop问题图形化定义的基础上,借鉴精英策略的思路,提出使用多种挥发方式的双向收敛蚁群算法,提高了算法的效率和可用性。最后,通过解决基准问题的实验,比较了双向收敛蚁群和蚁群算法的性能。实验结果表明,在不明显影响时间、空间复杂度的情况下,双向收敛蚁群算法可以加快收敛速度。

关键词:作业车间调度;蚁群算法;双向收敛
中图分类号:TP301 **文献标识码**:A

0 引言

如何合理高效地调度资源是影响模型效率的重要因素,长期以来一直受到广泛关注[1,2]。为了解决传统的作业车间调度(job-shop scheduling)问题,迄今已提出很多方法。文献[3]中使用Petri网和L1算法(A*算法的一种改进)结合求解;文献[4]中使用基于遗传的优化调度算法得到了较优的调度结果;文献[5]提出了一种多对数的估计算法,减少了求解时间。但这些算法不能保证取得问题的最优解,甚至是较优解。为此,本文提出了一种用于Job-Shop问题的双向反馈蚁群算法(Ant Colony Optimization,ACO),与传统ACO算法相比,明显提高了性能。

1 问题描述

Job-Shop问题既是实际生产中的一个重要问题,也是一个NP难题。该问题的特征模型如下:①存在j个工作(job)和m个机器(machine);②每个工作由一系列操作(或者任务/task/operation)组成;③操作的执行次序遵循严格的串行顺序;④在特定时间,每个操作需要一个特定机器完成;⑤每台机器在同一时刻不能同时完成不同的工作;⑥同一时刻,同一工作的各个操作不能并发执行;⑦问题是如何求得从第一个操作开始到最后一个操作结束的最小时间间隔(makespan)。

为了便于比较算法性能,本文采用Muth和Thompson在1963年提出的Job-Shop6×6基准问题,如表1所示。

表1 Muth&Thompson 6×6基准问题

	m,t	m,t	m,t	m,t	m,t	m,t
Job_1	3,1	1,3	2,6	4,7	6,3	5,6
Job_2	2,8	3,5	5,10	6,10	1,10	4,4
Job_3	3,5	4,4	6,8	1,9	2,1	5,7
Job_4	2,5	1,5	3,5	4,3	5,8	6,9
Job_5	3,9	2,3	5,5	6,4	1,3	4,1
Job_6	2,3	4,3	6,9	1,10	5,4	3,1

注:表中t表示任务所需时间。

该例中,Job_1的第一个任务(即上文中的操作,下同)$Task_1$需在$Machine_3$上完成,历时1个时间单位;第二个任务$Task_2$在$Machine_1$上完成,历时3个时间单位,以此类推。Job中的后续任务不能在

收稿日期:2003-06-27;**修订日期**:2003-10-10。
基金项目:国家863/CIMS主题资助项目(2001AA414610,2002AA414020,2002AA111080)。
作者简介:王常青(1978-),男,山东莱西人,中国科学院软件所智能工程实验室博士研究生,主要从事算法、业务流程管理、软件工程等方面的研究。E-mail:wcq@iel.iscas.ac.cn。

前面任务完成之前启动。求解 Job－Shop 问题,就是针对每一个 Machine,调度其上的任务次序。Muth&Thompson 问题的解如图 1 所示。

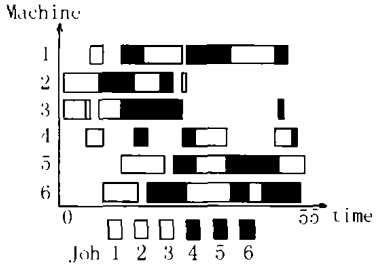

图1　Muth&Thompson问题的解

上述问题使用矩阵表示为:

$$\boldsymbol{T}=\begin{pmatrix} m_3 & m_1 & m_2 & m_4 & m_6 & m_5 \\ m_2 & m_3 & m_5 & m_6 & m_1 & m_4 \\ m_3 & m_4 & m_6 & m_1 & m_2 & m_5 \\ m_2 & m_1 & m_3 & m_4 & m_5 & m_6 \\ m_3 & m_2 & m_5 & m_6 & m_1 & m_4 \\ m_2 & m_4 & m_6 & m_1 & m_5 & m_3 \end{pmatrix},$$

$$\boldsymbol{P}=\begin{pmatrix} t_1 & t_3 & t_6 & t_7 & t_3 & t_6 \\ t_8 & t_5 & t_{10} & t_{10} & t_{10} & t_4 \\ t_5 & t_4 & t_8 & t_9 & t_1 & t_7 \\ t_5 & t_5 & t_5 & t_3 & t_8 & t_9 \\ t_9 & t_3 & t_5 & t_4 & t_3 & t_1 \\ t_3 & t_3 & t_9 & t_{10} & t_4 & t_1 \end{pmatrix}。$$

其中,矩阵 $\boldsymbol{T}$ 表示每个工作的任务调度顺序,矩阵 $\boldsymbol{P}$ 表示相应的时间间隔。采用文献[6]中的定义方法,将以上问题转换成如图 2 所示的结构。

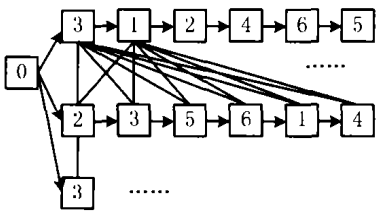

图2　Job-Shop问题的图形化定义

图 2 所示的结构由 37 个节点组成,增加了一个虚拟起点 0,从点 0 开始可以提供通往 Job_1,Job_2,…,Job_6 的单向通路,此外的节点 $P_{i,j}$ $(1\leqslant i,j\leqslant 6)$ 表示 Machine 矩阵 $\boldsymbol{T}$ 中相应位置的点,即 $\boldsymbol{T}_{ij}$ 中的 Machine 代号。因此,图 2 中的第 i 行(除起始点 0 以外)代表表 1 中的第 i 个 Job。每一个工作内的各个任务由有向弧连接,各个工作之间的任务由无向弧连接。每条路径定义值 τ_{ij} 表示从节点 i 到节点 j 路径上的信息素数量。

2　双向收敛蚁群算法

2.1　蚁群算法

蚁群算法是 Dorigo 在 1996 年提出的[6]。如图 3 所示,蚂蚁在离开巢穴寻找食物的过程中,每一只蚂蚁都会在自己身后留下一定量的信息素(pheromone),用于提示自己回巢的路径,并为其他蚂蚁提示食物的方向。信息素是一种易挥发的化学物质,由于蚂蚁经过较短路径时可以在较短时间内走完全程,相同时间内较短路径上可以积累较多信息素。以后出发的蚂蚁将倾向于选择信息素比较浓的路径。大量蚂蚁经过一定时间后将集中选择较短的路径。

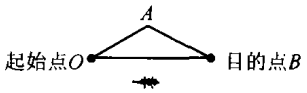

图3　蚂蚁寻径方法示意图

在图 3 中,假设两只蚂蚁从巢穴 O 出发到目的地 B 寻找食物,蚂蚁 m 沿 OB 行走,另一只蚂蚁 n 沿 OAB 行走,在 m 已经回到巢穴 O 时,n 还在路上,此时路径 OB 上的信息素浓度高于 OAB,因此后续的蚂蚁将较多选择路径 OB 行走。

2.2　相关定义

为使图 1 所示结构便于应用在蚁群算法中,对图 2 所示的图结构作如下定义:

定义 1　操作(Operation)用图 1 中的节点表示,定义操作为:

Operation=〈sTime, eTime, isDone, NEXT〉

其中,sTime 和 eTime 分别是本操作的开始和结束时间;isDone 表示本节点代表的操作是否已经完成,或者蚂蚁是否已经走过节点;NEXT 表示下一步的结构,包含了从本节点能够到达的节点表和到达这些节点的路径上信息素的数量表。

定义 2　图(Graph)为操作的集合:

Graph=〈Operation, Machine〉

Machine 为表 1 中表示操作所占用的机器的矩阵。在 Operation 节点中已经包含节点之间的连接,因此不需要定义节点之间的关系。

定义 3　虚拟起始点 startPoint 为:

startPoint=〈NEXT, nextMachine〉

虚拟起始点只连接每个工作的第一个操作。操作所占用的时间从 NEXT 结构中的 Operation 体现,因此虚拟起始点不占用操作时间。

2.3 算法描述

通过2.1中的分析可以看出，ACO算法的起始阶段不能迅速找到最优解的原因是不能确定最优解的位置，同时不能迅速排除明显较差的解。文献[6]中使用精英策略(elitist strategy)记录，并且只对历史最优解做奖励，能够明显加快算法收敛的速度。与此类似，本文提出了双向收敛策略，将历史最差解看作目前不可接受的解，对其进行惩罚，可以引导其他蚂蚁尽量远离历史最差解，放弃将最差解的组成部分组合成其他解的机会，从而加速算法的收敛。

传统ACO中挥发系数ρ可以促进信息素的挥发。实验证明[7]，没有挥发的ACO算法将很难得到即使是局部的最优解。挥发减少了所有路径上的信息素，在蚂蚁代数较大的情况下，能够使算法收敛于较好的解。同一个实验也证明，算法在开始阶段的很长一段时间内不能迅速避开较差的解，需要大量循环后才能逐步趋于优化。因此，有必要加强最差解方向上的惩罚，避免围绕最差解而产生大量干扰解，从而能够在算法不明显影响时间、空间复杂度的情况下提高算法性能。

以下是算法的描述：

步骤1 生成一代蚂蚁。根据算法规定的数量放出蚂蚁，为保证操作符合工作的调度顺序，在每只蚂蚁寻找路径的过程中，首先判断目的节点的前驱是否已经完成，在前驱已经完成并且本身尚未完成的所有节点中，使用信息素的浓度作为概率选取下一步目标。在目标的选取过程中，借鉴遗传算法中常见的轮盘方法(roulette wheel)决定。为保证蚂蚁遍历的次序符合Job操作的次序要求，使用下列原则：①Job中同一行的节点完成后不能直接转向自己的非直接后继点；②使用评价函数计算经过的路径代表的时间间隔Makespan时，遵守任务的先后次序，使蚂蚁行走时路径不代表任务次序；③次序的含义在计算Makespan时被加到路径上。

ACO算法在这一步需要对每只蚂蚁生成从当前步骤禁止访问的点，而在本算法中每只蚂蚁寻径时，无需每走出一步都对下一步的机器占用和工序次序做判断，大大降低了计算量，提高了算法的效率。同时，算法在计算路径转换可能性的过程中，不按照传统的方法将路径长度考虑在内。这是由于传统上单纯使用路径长度作为因子，导致完成时间较长的操作在蚂蚁寻径过程中处于不利地位，影响了算法的正确性。

假定蚂蚁行走的过程中不会重复已经走过的路径，蚂蚁选择下一条可能路径的状态转换规则是：

$$p_{ij}(t)=\frac{\tau_{ij}(t)}{\sum_{j\in \text{allowed nodes}}\tau_{ij}(t)}。$$

其中，$p_{ij}(t)$为蚂蚁选择从点i至点j之间道路的概率，$\tau_{ij}(t)$为点i至点j之间道路上的信息素数量，j属于从点i所允许访问的所有点集合。

步骤2 评价和激励。当本代蚁群所有蚂蚁完成对图结构中所有点的遍历后，使用评价函数f对得到的所有路径进行评价，并按以下规则确定信息素数量：

$$\tau_{ij}(t+\Delta t)=(1-\rho)\tau_{ij}(t)+\Delta\tau_{ij}(t+\Delta t)。\quad(2)$$

初始状态的信息素数量随机给出：

$$\Delta\tau_{ij}(t+\Delta t)=\begin{cases}\dfrac{Q}{f(\text{bestRoad})} & t+n\text{过程中的最优值}\\ -\dfrac{Q'}{f(\text{worstRoad})} & t+n\text{过程中的最差值}\\ 0 & \text{其他情况}\end{cases}。\quad(3)$$

式(2)中第一部分表示每一代蚂蚁走完全程后所有信息素挥发，ρ是挥发系数；第二部分表示信息素的修改，即惩罚函数，具体计算方法在式(3)中给出，取得迄今为止最好和最差的路径，对其进行惩罚。Q是单位路径上的信息素数量，Q'是单位路径上的用于惩罚最差值的信息素数量，$f()$是路径评价函数，计算方法将在2.5节中给出。

传统ACO使用参数调节的方法避免算法陷入局部最优，这种方法取决于具体的参数数值，往往导致一套参数对应于一个具体问题，降低了算法的通用性。本算法从两个方向进行反馈，很大程度上避免了对参数的依赖，同时加快了算法的收敛。

步骤3 循环执行。如果已经收敛于最优值或者到达最大蚂蚁代数，退出循环并返回算法结果；否则，循环执行步骤1、步骤2。

2.4 算法的时空复杂度分析

考虑到Job－Shop问题的图形化表述，本文建立一个n维数组和一个$n\times m$的矩阵表示问题空间。由于图2中无向弧表示的两个方向上的信息素并不一致，在矩阵内部使用二维结构表征两个不同的路径上的信息量。因此，可以得到算法的空间复杂度：

$$O((m\times n)\times(m\times n))。$$

由于算法需要对最优和最差两方面的解作出响应，所以算法需要增加两个大小为$m\times n$的数组以保存路径，但是这并不影响算法的空间复杂度。

算法使用多代(numOfCycle)蚂蚁,每一代有一定数目(numOfAnt)的蚂蚁行走。对同一只蚂蚁而言,每个点只经过一次,对最优解和最差解的处理过程包含于每一代蚂蚁行走结束以后,增加的复杂度仅仅与蚂蚁代数有关,因此不会明显影响算法的时间复杂度,算法的时间复杂度与蚂蚁和节点的数量成线性关系:

$O(\text{numOfCycle}\times\text{numOfAnt}\times(m\times n))$。

2.5 评价函数

评价函数的基本思路是,规定每一只蚂蚁经过的路径中,前面节点的开始时间不会落在后面节点之后。由于蚂蚁在寻径过程中已经考虑到了操作在工作中的先后次序,结果中的节点串将表征不同工作的时间次序。具体算法如下:

算法 1　求时间间隔 Makespan

```
//初始化第一个操作的时间
O1.startTime=0;
O1.endTime=O1.executeTime;
MO1.Time=O1.endTime;
Makespan=0;
for(int I=2;I<=m*n;I++)//根据蚂蚁的路径找到相应操作的起止时间
{
  OI.startTime=OI-1.endTime;//本次操作只能在前面操作已经完成的情况下执行
  找到 OI 对应的机器 MOI;
  if(MOI.Time> OI.startTime) then//如果这时机器被占用
  OI.startTime=MOI.Time;//等机器释放后开始
  OI.endTime=OI.startTime+OI.executeTime;
  MOI.Time=OI.endTime;
  }
for(int j=1;j<=NumberOfMachine;j++)//求出完成所有工作所需时间
{
  if Makespan<Mj.Time then
    Makespan=Mj.Time;
}
```

使用该算法可以求出每只蚂蚁遍历图的所有节点之后得出的 Job－Shop 问题的解。根据这个解,对所有蚂蚁路径中最优的一条和最差的一条使用式(3)改变信息素。文献[8]表明,相对于更新所有路径上的信息素,只对最优路径做处理的方式可以得到更好的进化特性。

3　实验结果及分析

在 PIV 1.5G/128M 系统中使用 VC++6.0 完成算法,计算 Muth&Thompson 的 6×6 基准问题。实验结果如表 2 所示,该结果说明了在现有算法基础中,挥发系数对系统收敛的影响,挥发系数过大时,有效路径上的信息素迅速减少至 0,导致算法在局部最优点收敛;挥发系数过小时,由于较优路径上遗留的信息素相对于较差路径不具有优势,较优的路径得不到相应的激励,算法仍然不能在最大循环次数之内收敛。

表 2　挥发系数对收敛结果的影响

蚂蚁数目	循环次数	ρ	收敛结果
200	500	0.005	57.6
200	500	0.010	56.4
200	500	0.015	56.0
200	500	0.017	55.0
200	500	0.020	57.8

注:取 5 次结果平均值

表 3 为蚂蚁数目对收敛结果和耗时的影响,从中可以看出,在循环次数小于 500、蚂蚁数目为 200 时能够顺利收敛至最优解,因此用时也最少。

表 3　蚂蚁数目对收敛结果和耗时的影响

蚂蚁数目	最大循环次数	ρ	收敛结果	所用时间/秒
36	500	0.017	59.0	14
72	500	0.017	57.6	29
100	500	0.017	57.8	41
150	500	0.017	56.8	61
200	500	0.017	55.0	23
250	500	0.017	56.8	100
300	500	0.017	58	124

注:取 5 次结果平均值

表 4 比较了双向收敛 ACO 和传统 ACO 算法所得到的解。双向收敛 ACO 使用了较多蚂蚁进行并行搜索,提高了搜索过程的挥发系数,从而在较少的代数中得到了问题的解。实验证明,双向收敛 ACO 算法相对于传统 ACO 算法,所用蚂蚁总数减少 42.4%,大幅度提高了算法的效率。

表 4　双向收敛 ACO 和 ACO 最优解的比较

算法	蚂蚁数目	循环次数	蚂蚁总数	ρ	收敛结果
传统 ACO[8]	36	3000	108000	0.010	55
双向收敛 ACO	200	311	62200	0.017	55

注:5 次结果平均值

4　结束语

本文使用了双向收敛 ACO 算法解决 Job－

Shop问题,试验结果证明这种方法是行之有效的。在处理过程中,如果对挥发系数和信息素的调节不当,往往使算法陷入局部最小值。为避免这一问题,传统上采用的方法往往是降低信息素的挥发以保存有益的信息素,或者调节惩罚函数的参数以获得收敛,而本文则使用双向惩罚函数加快算法收敛,避免陷入局部最优,与传统方法相比较,明显提高了算法效率。本算法是针对Job-Shop问题的,但也可以很容易地使用在其他可归纳为图结构的调度问题上,具有很好的应用价值。

参考文献:

[1] CLEVELAND G A, SMITH S F. Using genetic algorithms to schedule flow shop release[A]. Proceedings of the 3rd International Conference on Genetic Algorithms(ICGA)[C]. San Mateo: Morgan Kaufmann Publishers, Inc., 1989. 160-169.

[2] JAIN A S, MEERAN S. Deterministic job-shop scheduling: past, present and future[J]. European Journal of Operational Research, 1999, 113(2): 390-434.

[3] LEE D Y, DICESARE F. Scheduling flexible manufacturing systems using Petri nets and heuristic search[J]. IEEE Transaction on Robotics and Automation, 1994, 10(2): 123-132.

[4] JIANG Sijie, XU Xiaofei, LI Quanlong. An optimal algorithm for a class of Jobshop scheduling problems[J]. Computer Integrated Manufacturing Systems, 2002, 8(3): 229-232(in Chinese). [姜思杰,徐晓飞,李全龙. 基于遗传优化算法求解作业车间调度问题[J]. 计算机集成制造系统, 2002, 8(3): 229-232.]

[5] SHMOYS D B, STEIN C J W. Improved approximation algorithms for shop scheduling problems[J]. SIAM Journal on Computing, 1994, 23: 617-632.

[6] DORIGO M, MANIEZZO V, COLORNI A. The ant system: optimization by a colony of cooperating agents[J]. IEEE Transactions on Systems, Man and Cybernetics - Part B, 1996, 26(1): 29-41.

[7] DORIGO M, STÜTZLE T. An experimental study of the simple ant colony optimization algorithm[A]. WSES International Conference on Evolutionary Computation (EC'01)[C]. Athens: WSES-Press, 2001. 253-258.

[8] ZWAAN van der S, MARQUES C. Ant colony optimization for job shop scheduling[A]. Proceedings of the Third Workshop on Genetic Algorithms and Artificial Life (GAAL 99)[C]. 1999.

Bi-directional convergence ACO for job-shop scheduling

WANG Chang-qing, CAO Yun-fu, DAI Guo-zhong

(Inst. of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: To properly and efficiently schedule resources and solve the combinatorial optimization problem, an improved algorithm named Bi-directional Convergence Ant Colony Optimization (ACO) algorithm was proposed. Using the graphic definition of Job-shop problem and the elitist strategy, the Bi-directional Convergence ACO algorithm was designed to improve efficiency and usability of original ACO by different evaporated means. Finally, the Bi-directional Convergence ACO algorithm was tested on a benchmark Job-shop scheduling problem. The performance of the Bi-directional Convergence ACO was also compared with that of the original ACO. The simulation result illustrates that the bi-directional convergence ACO algorithm accelerates the convergence without affecting the temporal and spatial complexity much.

Key words: job-shop scheduling; ant colony optimization algorithm; bi-directional convergence

Received 27 Jun. 2003; accepted 10 Oct. 2003.

Foundation item: Project supported by the National High-Tech. R&D Program, China (No. 2001AA414610, 2002AA414020, 2002AA111080).