基于遗传算法的 Job-Shop 调度问题求解方法*

陈恩红 刘贵全 蔡庆生

(中国科学技术大学计算机系 合肥 230027)

E-mail: csehchen@comp.polyu.edu.cn

摘要 调度问题是许多计算机应用领域的重要问题,Job-Shop 调度是其中的一类典型的困难问题,它通常包含多个可并行实现的目标以及实现这些目标的多种方法与资源.本文以一类实用的 Job-Shop 问题模型为基础,给出了用遗传算法求解调度问题应采用的染色体表示方法,并针对问题的特点,给出了面向资源空间与面向规划空间的遗传操作的设计思想与方法.实验结果表明,基于遗传算法的 Job-Shop 调度问题求解方法具有较好的性能,同时也表明,对于求解过程中可能出现的提前收敛问题可通过改变遗传操作概率及调节适应度等方法予以解决.

关键词 Job-Shop 调度,遗传算法.

中图法分类号 TP18

如何有效地解决调度问题是许多计算机应用领域面临的一个困难而重要的问题.[1~4]典型的调度问题包含多个可并行实现的目标以及实现这些目标的多种方法及资源.目标、方法及资源的组合是问题的状态空间呈指数形式增长.作为调度问题中代表性的 Job-Shop 调度,其目标是以尽可能少的时间,同时满足其它一些约束条件情况下,将各种操作调度到适当的机器上,分别加工某些构件,最终生产出某种产品.下面是一个反映调度问题一般特征的模型[4]:

- 每种 Job 生产一种产品;
- 每种产品由多个构件组成,每种构件数量若干;
- 一个 Job-Shop 可生产多个构件;
- 每种构件有多种可供选择的生产规划;
- 每种规划由一系列串行操作构成;
- 每种操作需要若干资源,如机器等(本文仅考虑这一种);
- 每台机器可用于不同类型的操作,每种操作可用机器若干台;
- 每种操作需占用某机器一段时间;
- 每种 Job 由一个需求序列组成;
- 每种需求对应一个构件,即一个产品有多少构件,Job 中就有多少种需求.

对于某些问题可能还有其它约束.求解这类调度问题就是设法将生产各种构件的操作合理地分配到可供使用的机器上,以求尽可能地提高机器的利用率和构件响应速度.

与上述问题模型相关的搜索空间有:需求空间,即各种构件的生产顺序;规划空间,生产每种构件可采用若干种不同的规划实现;资源空间,每种规划中的每种操作都可能在若干种不同的机器上实现.

由于每种规划和分配给它的资源联系紧密,所以往往将资源与规划空间作为一个整体来考虑.每种规划实现的可能方法数为规划中每个操作可选择的资源数的乘积,每种构件生产的方法数为生产该构件的规划数之和,由此可知总的规划、资源空间大小可表示如下:

$$\prod_{i=1}^{n\_parts} \sum_{j=1}^{n\_plan[i]} \prod_{k=1}^{n\_op[i,j]} n\_resource[i,j,k]$$

---

* 本文研究得到国家自然科学基金、国家教委博士点基金和中国科学技术大学青年基金资助.作者陈恩红,1968年生,博士,讲师,主要研究领域为遗传算法,机器学习,约束满足问题.刘贵全,1970年生,博士生,主要研究领域为机器学习,知识发现.蔡庆生,1938年生,教授,博士导师,主要研究领域为人工智能,机器学习.

本文通讯联系人:陈恩红,合肥 230027,中国科学技术大学计算机系

本文 1997-01-20 收到原稿,1997-04-03 收到修改稿

其中 n_part 为构件的类数，n_plan[i]表示生产构件 i 的可能规划数，n_op[i,j]表示生产构件的第 j 种规划所含操作数，n_resource[i,j,k]为生产构件 i 的第 j 种规划中的第 k 个操作可用的资源数.

当一种 Job 由多种需求组成，每种构件可用多种规划实现，每类操作可选用多种资源时，问题的搜索空间就会呈指数形式增长. 由于时间与空间的限制，盲目搜索算法无力求解这样复杂的调度问题，而启发式搜索方法在搜索空间太大时也难以奏效. 采用遗传算法[5~7]可在很大的状态空间中随机高效地采样、搜索，并较快地收敛到最优或近似最优解. 本文将详细给出基于遗传算法的 Job-Shop 调度实现方法，最后对该方法的性能进行分析.

1 基于遗传算法的 Job-Shop 调度实现

1.1 染色体的表示方法

用遗传算法求解 Job-Shop 调度问题，首先要为每种 Job 的调度寻求一种适当的表示方法. 通常，染色体表示应遵循两条原则：染色体中每种基因与问题相关而与其它基因无关；选取最小字母表表示问题.

对于该问题，一种可能的方法是以每台机器的调度来构造染色体，每台机器的调度用分配到该机器的各种操作表示，操作按时间顺序排列. 采用这种表示容易实现染色体适应度的计算，且无须调度生成过程的介入，但相应的遗传操作设计非常复杂. 为了充分利用遗传算法的优点，在 Job-Shop 调度问题中，将资源及规划两种空间都纳入遗传算法的搜索范围，本文采用如下形式的表示：

or_2	or_1	or_2
$op_1\ op_2\ op_7$	$op_1\ op_4\ op_6$	$op_3\ op_5\ op_6$
$M_1\ M_1\ M_2$	$M_1\ M_1\ M_2$	$M_1\ M_3\ M_3$

其中 or_i 为第 i 种需求；op_j 为实现相应需求的操作；M_1，M_2，M_3 为操作所需的机器. 染色体表示所需数据结构定义如下：

```
struct op_machine
  { unsigned char op;
    unsigned char mach; }
struct order
  {
    unsigned char order_num; /* 定义需求号 */
    unsigned char plan_num; } /* 定义实现需求 order_num 的规划号 */
    struct op_machine op_mach[PLAN_LENGTH]
    /* 定义一种规划，PLAN_LENGTH 为规划长度即操作数目. */
  }
struct individual
  {
    struct order or[CHROM_LENGTH];
    /* 定义一个染色体，CHROM_LENGTH 为染色体中需求数 */
    float fitness; /* 定义染色体的适应度 */
  } newpop[POP_SIZE],oldpop[POP_SIZE], *p_oldpop, *p_newpop;
    /* POP_SIZE 为群体规模 */
```

1.2 调度生成的实现方法

由于采用的染色体表示较复杂，因此需设计出将染色体转换成适当调度的方法. 为了生成一个调度，需记录每台机器的调度情况. 由于调度到每台机器上的操作经常变动，且数目无法预先确定，因此，本文采用如下数据结构记录机器的调度情况：

```
struct schedule
  { int op;
    float s_time; /* 操作的起始时间 */
    float e_time; /* 操作的终止时间 */
    struct schedule *next; };
```

每台机器需一个链表，表头指针组成一个数组：

```
struct schedule *machine_schedule[TOTAL_MACHINE].
```

为便于染色体适应度值的计算，还需纪录染色体中每个规划的开始时间和终止时间：

```
float plan_start_t[CHROM_LEN],f_plan_finish_t[CHROM_LEN];
```

由此,可方便地实现由染色体 i 生成相应调度的方法,其过程如下:

1)初始化相应于 i 的机器调度链表*machine_schedule,并从染色体的第1个规划or[l].plan_num开始处理;

2)检查 i 中所有规划是否处理完;若是,调度生成过程结束,并返回最后规划的结束时间;

3)检查or[k].op_mach[m]中的 m 是否为1,若是,则以机器or[k].op_mach[m].mach上的上一操作的结束时间为or[k].op_mach[m].op的开始时间,同时它也是当前规划的开始时间;否则,取该机器上前一操作结束时间和or[k].op_mach[$m-1$].op的结束时间的较大值为操作or[k].op_mach[m].op的开始时间,该操作的结束时间为其开始时间加所需占用的时间;

4)检查or[k].op_mach[m]中的 m 是否为PLAN_LENGTH,即第 k 个规划的最后一个操作是否已处理完,若是,则记录规划结束时间,转步骤7;

5)将该操作调度加入到它所分配的机器or[k].op_mach[m].mach的链表*machine_schedule上;

6)$m \leftarrow m+1$;转步骤3,处理下一操作or[k].op_mach[m];

7)$k \leftarrow k+1$;转步骤2,处理 i 的下一规划.

1.3 适应度函数的设计

得到染色体对应的调度后,可以较方便地得到其适应度值.为了反映机器的利用率及各个需求的响应时间(不同需求可以赋予不同的优先级),本文采用了如下方式来定义适应度函数:

$$f(i) = \mu_1 \cdot f_1(i) + \mu_2 \cdot f_2(i),$$

其中 $\mu_1+\mu_2=1$;

$f_1(i)=\sum_{j=0}^{m-1}\sum_{k=0}^{n-1}$machine[buffer.op].time[buffer.mach] / (K * 调度持续时间);

$f_2(i)=\sum_{j=0}^{m-1}$(W[j] · $\sum_{k=0}^{n-1}$machine[buffer.op].time[buffer.mach] / (K * 第 m 个规划持续时间));

m=CHROM_LENGTH;

n=PLAN_LENGTH;

buffer=newpop[i].order[j].op_mach[k].

W[j]≤1为第 i 种规划实现的需求在总需求中的优先级,若所有W[j]均为1,则不考虑需求间的优先级.计算适应度时,也可以实现某些其它形式的约束,并对违反约束条件的调度进行严格的惩罚,即锐减其适应度.

1.4 遗传操作设计

为了利用具体领域信息,必须定义适于Job-shop问题本身的遗传操作,主要是交叉操作和变异操作.

(1) 交叉操作

由于染色体包含了资源及规划信息,因此,实现交叉操作时可考虑两类不同方法,分别描述如下:

· 面向资源空间的交叉操作

随机选取两个染色体 i,j,其中 i 由规划序列 $p_{i1},p_{i2},\ldots,p_{in}$ 组成,j 由规划序列 $p_{j1},p_{j2},\ldots,p_{jn}$ 组成,随机选取两交叉点 k,l,满足 $1\leqslant k\leqslant l\leqslant n$,若 i 的子序列 $p_{i,k},p_{i,k+1},\ldots,p_{i,l}$ 中某一规划 p_{i,m_i}($k\leqslant m_i\leqslant l$)与 $p_{j,k},p_{j,k+1},\ldots,p_{j,l}$ 中规划 p_{j,m_j} 为同一类型规划(即实现相同需求),交换 p_{i,m_i} 与 p_{j,m_j} 中相同操作所用资源(机器).

· 面向规划空间的交叉操作

对于这种类型的交叉操作有两种具体形式,分别完成对同一规划集不同序列的搜索以及对同一类(实现同一种需求)的规划的搜索,即搜索生产同一构件的不同操作序列及相应机器.

随机选取染色体 i,j,其中 $i=p_{i1},p_{i2},\ldots,p_{in}$,$j=p_{j1},p_{j2},\ldots,p_{jn}$,随机选则两交叉点 k,l,其中 $1\leqslant k\leqslant l\leqslant n$.

第1种交叉操作的方式如下:

对 i 的子序列 $p_{i,k},p_{i,k+1},\ldots,p_{i,l}$ 中每一规划 $p_{i,m}$,均先找出 j 的子序列 $p_{j,k},p_{j,k+1},\ldots,p_{j,l}$ 中位置与之对应的 $p_{j,m}$,再根据 $p_{j,m}$ 在 i 中找出实现相同需求的规划 $p_{i,m'}$,其中 $m'\in\{1,2,\ldots,n\}$ 且 $m'\neq m$,并交换 $p_{i,n}$ 与 $p_{i,m'}$,从而达到对不同规划序列搜索的目的,这种交换共进行 $l-k+1$ 次.

第2种交叉操作的方式如下:

首先对 i 中交叉点间的子序列 $p_{i,k},p_{i,k+1},\ldots,p_{i,l}$ 中每一规划 $p_{i,m}$($k\leqslant m\leqslant l$),在 j 中找出与之实现同类需求的规划 $p_{j,m'}$,其中 $m'\in\{1,2,\ldots,n\}$,然后再交换 i 中 $p_{i,m}$ 与 j 中 $p_{j,m'}$,从而完成对实现同类需求的规划的搜索.

(2) 变异操作

与交叉操作对应，也有两类变异操作：

• 面向资源空间的变异操作

设变异前，染色体 i 中规划序列为 $p_{i1},p_{i2},\ldots,p_{in}$，变异后为 $p_{i1}',p_{i2}',\ldots,p_{in}'$，经过该类变异操作后，$p_{i,j}$与 $p_{i,j}'$($j=1,2,\ldots,n$)满足如下关系：

$$op_k(p_{ij}')=op_k(p_{ij}),k=1,2,\ldots,NUM\_OF\_OP(p_{ij})$$

且

$$M(op_k(p_{ij}'))\begin{cases}=M(op_k(p_{ij})),\theta_i>p_m\\ \in M\_SET(op_k(p_{ij}))\wedge\neq M(op_k(p_{ij})),\theta_i\leqslant p_m\end{cases}$$

其中 $op_k(p_{ij})$为组成规划 $p_{i,j}$的操作序列中的第 k 个操作；$NUM_OF_OP(p_{ij})$表示规划 $p_{i,j}$中操作总数；$M_SET(op_k(p_{ij}))$表示 $p_{i,j}$的第 k 个操作可能使用的机器集；$M(op_k(p_{ij}))$表示 $p_{i,j}$的第 k 个操作使用的机器；p_m 为变异概率，θ_i 是 0 与 1 之间的随机数，对于染色体中每一规划的每一操作都要重新采样.

• 面向规划空间的变异操作

这种类型的变异操作有两种具体形式：

对第 1 种形式的变异操作，p_{ij}与 p_{ij}'满足如下关系：

$$p_{ij}'=\begin{cases}p_{ij}',\theta_i>p_m\\ p_{ix}',\theta_i\leqslant p_m\end{cases}$$

对第 2 种形式的变异操作，p_{ij}与 p_{ij}'满足如下关系：

$$p_{ij}'=\begin{cases}p_{ij}',\theta_i>p_m\\ P\_SET(p_{ij})\wedge\neq p_{ij})),\theta_i\leqslant p_m\end{cases}$$

且 p_{ij}'中每种操作使用的机器从其可用的机器集中任选. 其中 $p_SET(p_{i,j})$为与 $p_{i,j}$实现相同需求的规划集.

1.5 遗传算法实现

至此，本文已详细介绍了遗传算法求解 Job-shop 调度问题的染色体表示方法、适应度函数的设计思想以及各种交叉、变异操作. 为简明起见，本文将我们的遗传算法实现总结为以下几个关键步骤：

```
step1: initialize();          /* 生成初始群体 */
step2: statistic();           /* 对每个染色体构造调度，并计算适应度值 */
step3: scaling();             /* 调节适应度值 */
step4: reprcduction();        /* 根据调节后的适应度值，选择染色体进行复制 */
step5: generation();          /* 进行交叉、变异操作生成下一代群体 */
step6: statistic();
step7: change_prob();         /* 若改变交叉、变异操作概率条件成立，则改变 */
step8: if 算法终止条件成立 then 输出当前最优解并结束 else step3.
```

2 算法性能分析

表 1 反映了几种因素对算法性能的影响，其中的平均适应度值是指群体随机初始化后，群体所有的染色体的平均适应度值，已知最佳解是算法的各次运行中发现的最好的解，并不是指问题的实际最优解.

问题模型中 Job-Shop 设置如下：3 台机器，3 种构件，每种构件数量 3～4 个，每种构件可选用 3 种不同规划生产，每种规划由 2～3 个动作序列组成，每种操作可调度到 2～3 台机器上，共有 10 种需求. 算法中使用的交叉及变异操作的初始概率分别为 $P_c=0.4$，$P_m=0.0015$.

其中 case1 指算法运行中既不改变交叉、变异操作概率，也不对适应度值进行调节时的情况，case2 指算法运行 500 代后，让交叉总概率 P_c 增为 0.6，变异总概率 P_m 增为 0.003 的情况，case3 指算法对染色体适应度值进行调节，使最大适应度值是平均适应度值的 2 倍的情况，case4 指算法运行中对染色体适应度值进行调节，同时在运行 500 代后，让交叉总概率 P_c 增为 0.6，变异总概率 P_m 增为 0.003 的情况. 由表 1 可见，当群体性能改进不大时改变遗传操作的使用概率，对算法性能有所提高；调节适应度值改善了算法的收敛性. 对该问题，开始运行时，调节适应度值没有起到抑制作用，这是因为最大适应度值没有超过初始群体平均适应度值的 2 倍. 实际上，这种情况下不需要抑制，因此可在运行若干代后才调节适应度值.

表1 算法性能比较(每代适应度值均为10次运行结果的最大适应度平均值)

运行代数	适应度值			
	case1	case2	case3	case4
100	0.746	0.746	0.753	0.752
200	0.768	0.767	0.77	0.781
300	0.791	0.792	0.795	0.796
400	0.798	0.798	0.810	0.811
500	0.807	0.809	0.814	0.814
600	0.809	0.813	0.821	0.830
700	0.810	0.817	0.827	0.841
800	0.810	0.821	0.838	0.852
900	0.814	0.827	0.851	0.861
1000	0.814	0.835	0.857	0.864

3 结 论

用遗传算法求解 Job-Shop 调度问题的结果总体上说是较为有效的,这说明遗传算法是一种较好的搜索算法,但如果处理不当,容易在搜索过程中产生提前收敛到非全局最优解或非近似最优解的问题.为了避免产生这一问题,可采用改变交叉和变异概率、调节适应度值等方法.本文的工作主要针对 Job-Shop 调度问题的,但我们很容易将这种方法推广应用到其它类型的调度问题上.

参考文献

1 Smith S F, Fox M S, Ow P S. Constructing and maintaining detailed production plans: investigations into the development of knowledge-based factory scheduling systems. AI Magazine, 1986,7(4):45~61

2 Cleveland G A, Smith S F. Using genetic algorithms to schedule flow shop release. In: Proceedings of the 3rd International Conference on Genetic Algorithms. 1989. 160~169

3 Grefenstette J J. Incorporating problem-specific knowledge into genetic algorithms. In: Davis L ed. Genetic Algorithms and Simulated Annealing. 1987. 42~60

4 Uckun S, Bagchi S, Kawamura K. Managing genetic search in job shop scheduling. IEEE Expert, 1993. 15~24

5 Holland J H. Adaptation in natural and artificial systems. University of Michigan Press, 1975

6 Goldberg D E. Genetic algorithms in search, optimization and machine learning. Reading: Addison-Wesley, 1989

7 Davis L. Handbook of genetic algorithms. Van Norstrand Reinhold, New York, 1991

A Genetic Algorithm Based Job-Shop Scheduling Problem Solving Method

CHEN En-hong LIU Gui-quan CAI Qing-sheng

(*Department of Computer Science University of Science and Technology of China Hefei* 230027)

Abstract Scheduling is an important problem for many computer application areas. Job-Shop scheduling is a typical kind of difficult problems among them. It usually comprises several parallel goals, methods and resources available to realize the goals. Based on a practical Job-Shop problem model, the paper presents a chromosome representation method for genetic algorithm based on the scheduling problem. To fully use the knowledge of the problem, the authors also propose resource-space oriented and plan-space oriented genetic operators. The experiment result demonstrates that the performance of the method is satisfactory, and it also shows that the pre-mature problem can be solved by dynamically changing the probability of genetic operators or by scaling the fitness values of chromosomes.

Key words Job-Shop scheduling, genetic algorithm.

Class number TP18