文章编号：2096-1472(2022)-10-41-04
DOI:10.19644/j.cnki.issn2096-1472.2022.010.009

# 遗传算法在车间调度中的研究

李志林

(浙江理工大学理学院，浙江 杭州 310000)
✉zhilin_li19@163.com

**摘　要**：调度问题关系到车间生产的效率，是生产领域长期关注的问题。针对工件加工时需要满足额外资源约束的平行机车间调度问题，设计一种可行的排序，使得最大完工时间最小。采用遗传算法求解该模型，对种群的产生增加了可行性判定条件，并设置算法中的选择、交叉、变异等算子进行迭代，同时直接以目标函数作为适应度更利于搜索，利用Python 3.10.1进行了数值模拟实验，在随机产生的大量实例中，算法解与最优解下界的比值稳定在1.2以内。结果表明，文中的遗传算法对于资源约束的调度问题有很好的优化效果。

**关键词**：额外资源；平行机；数值模拟；遗传算法
**中图分类号**：TP39　**文献标识码**：A



# Research on Genetic Algorithm in Workshop Scheduling

LI Zhilin

( *Department of Science*, *Zhejiang Sci-Tech University*, *Hangzhou* 31000, *China*)
✉zhilin_li19@163.com

**Abstract:** Scheduling is a long-term concern in the production field as it has much to do with the efficiency of workshop production. Aiming at the workshop scheduling problem of the parallel machines that need to meet additional resource constraints during workpiece processing, this paper proposes to design a feasible ordering to minimize the maximum completion time. Genetic algorithm is used to solve the model, which adds the feasible judgment conditions to the generation of the population. The selection, crossover, mutation and other operators in the algorithm are set to iterate, and meanwhile, objective function is directly used as the fitness to facilitate the search. Python 3.10.1 is used for numerical simulation experiments. In a large number of randomly generated instances, the ratio of the lower bound of the algorithm solution and the optimal solution is stable within 1.2. Results show that the proposed genetic algorithm has a good optimization effect on scheduling problems with resource constraints.

**Keywords:** extra resource; parallel machine; numerical simulation; genetic algorithm

## 1 引 言(Introduction)

工件的加工需要资源[1]，资源也是优化目标的一个重要因素。资源的定义很广泛，例如加工工件所需的设备、员工、原材料等。我们把一般的资源，例如加工设备等统称为机器，把除了加工设备以外的其他资源称为额外资源。额外资源对调度的影响是多样化的，例如额外资源使用的数量往往可以决定工件的加工速度；额外资源有时需要不断损耗，有时又可以部分回收等。

遗传算法在函数优化、组合优化、生产调度以及机器学习领域有着广泛的应用。该算法基于对生物遗传和进化过程中选择、交叉、变异机理的模仿，来完成对问题最优解的自适应搜索过程。关于遗传算法在车间调度中的应用，陈金广等[2]以最小化最大完工时间为优化目标，初始化时将种群规模扩大以增加种群多样性，同时使用新的适应度函数让染色体

间更易区分。对于平行机调度的问题，孙超平[3]研究了一类考虑外包的平行机调度问题，目标是使作业外包总成本与最大完工时间同时最小化。

机器环境为 $m$ 台同类型的平行机上加工 $n$ 个具有额外资源工件的调度问题，目标函数为最小化最大完工时间 $\min C_{\max}$，每个工件加工时只需要一种额外资源，用 $P|res\lambda\sigma 1|C_{\max}$ 来表示。其中 $res\lambda\sigma 1$ 表示共有 $\lambda$ 种资源，每种资源同一时刻有 $\sigma$ 个单位可用，且每个工件加工只需要消耗1 个单位资源。

对于目标函数为极小化最大完工时间的有资源约束的平行机调度问题 $P|res\lambda\sigma\delta|C_{\max}$，该问题已经在很多文献中进行了研究[4−7]。特别地，当工件的加工时间均为单位时间时，GAREY等[5]证明问题 $P_2|res\cdots,p_i=1|C_{\max}$ 在多项式时间 $O(n^3)$ 内可解；EDIS等[7]在前人的基础上，介绍了在工件的加工时间都是单位时间的前提下，对于问题当工件具有到达时间情况下的模型 $P|res1\cdot 1,r_i,p_i=1|C_{\max}$、问题 $P|res\lambda\sigma\delta,\ p_i=1|C_{\max}$ 和问题 $P|res1\cdot 1,p_i=1|\sum C_j$ 都是多项式时间可解的。

本文研究问题表示为 $P_m|res\lambda 11|C_{\max}$，表示共有 $\lambda$ 种资源，每种资源同一时刻有1 个单位可用，且每个工件加工只需要消耗1 个单位资源，目标为最小化最大完工时间。本文针对该问题特点，设计遗传算法来求解，在产生初始种群和交叉变异种群中加入了可行性的判定条件，同时为了避免原种群的较优的个体丢失，每次迭代，融合新旧种群，选取融合种群中的最优解进行输出，并从融合种群中选取适应度高的个体作为下一次迭代的父代。最后,对本文设计遗传算法进行了数值模拟实验，利用Python 3.10.1来实现，实验结果表明，算法解与最优解下界的比值稳定在1.2以内，证明了遗传算法对所研究问题的有效性。

## 2 问题的符号介绍及相关描述(Symbol introduction of the problem and its related description)

### 2.1 $P_m|res\lambda 11|C_{\max}$的相关符号介绍

$m$：$m$ 台平行机($m$ 台一样的机器)；

$M_i$：第 $i$ 台加工的机器，$i=1,2,\cdots,m$；

$n$：待加工的工件数量；

$N=(J_1,J_2,\cdots,J_n)$：需要加工的工件集合；

$J_j$：代表第 $j$ 个工件，$j=1,2,\cdots,n$；

$p_j$：工件 $j$ 的加工时间，$j=1,2,\cdots,n$；

$\lambda$：可用资源种类数；

$N=\{N^1,N^2,\cdots,N^u,\cdots,N^\lambda\}$：不同资源的工件划分；

$p\left(N^\lambda\right)$：资源 $\lambda$ 对应的所有工件加工时间之和；

$C_i$：第 $i$ 台机器的完工时间；

$C_{\max}$：所有机器中的最大完工时间 $C_{\max}=\max(C_i)$，$i=1,2,\cdots,m$。

### 2.2 问题的相关描述

研究机器环境为 $m$ 台同类型的平行机上加工 $n$ 个具有额外资源工件的调度问题，目标函数为最小化最大完工时间 $\min C_{\max}$，每个工件加工时只需要一种额外资源，用 $P_m|res\lambda 11|C_{\max}$ 来表示。其中 $P_m|res\lambda 11|C_{\max}$ 表示共有 $\lambda$ 种资源，每种资源同一时刻有1 个单位可用，且每个工件加工只需要消耗1 个单位资源。

工件集合用 $N$ 来表示，同时将它划分为 $\lambda$ 个不同的子集 $N=\{N^1,N^2,\cdots,N^u,\cdots,N^\lambda\}$，其中 $N^u(u=1,2,\cdots,\lambda)$ 代表所需资源为 $u$ 的工件集合。$P\left(N^u\right)$ 为集合 $N^u$ 中所有工件的加工时间之和。若对于任意的资源 $u$，$N^u$ 中的工件个数都为1，此时可以不考虑资源冲突问题。则该问题可以归结为 $P\|C_{\max}$，是NP难的，即使是机器数为2。

问题的可行性条件：设有可行解的目标函数值 $C_{\max}$，为对于某一资源 $u$，在任意时刻 $t\in[0,C_{\max}]$，最多有1台机器正在使用。

令 $C^*_{\max}$ 为问题的最优解，则问题的最优解下界 $C_B$[7]满足：

$$C^*_{\max}\geqslant C_B=\max\left\{\frac{p(N)}{m},\max_{1\leqslant u\leqslant\lambda}p\left(N^u\right)\right\}$$

## 3 遗传算法的设计(Design of genetic algorithm)

遗传算法在函数优化、组合优化、生产调度以及机器学习领域有着广泛的应用。不同于其他调度问题，本文随机产生初始种群时，并不能够保证种群中的解是可行的，因此本文相对于其他遗传算法产生初始种群过程，多了可行性判定过程，初始种群的每个可行解都要满足本文资源约束的条件。这对于算法的运行时间有着重要的影响。同时在每次产生交叉种群与变异种群的过程中，也加入判别可行性的步骤，选取满足条件的解加入新种群，每次迭代，选取种群中最优的解进行输出。

通过上述性质可知，遗传算法以群体搜索为特性，优化结果与初始条件无关，这将使得优化结果更好。下面给出具体的遗传算法设计。

### 3.1 编码方式

在实际应用中，遗传算法的编码方式直接影响算法的执行效率，针对本文问题的特点，设计如下编码方式：每个解由一个长度为 $n$ 的整数串 $(a_1,a_2,\cdots,a_n)$ 组成，每个串位 $a_i$ 满足 $1\leqslant a_i\leqslant m\ (1\leqslant i\leqslant n)$ 表示对应的 $n$ 个工件分别在哪台机器上加工，同时，相同机器在该编码中出现的顺序，代表该位置工件在该机器上加工的实际顺序。

例如：对工件集合 $N=(J_1,J_2,J_3,J_4,J_5,J_6,J_7)$，机器为 $M_1,M_2,M_3$ 的实例，如若其编码串表述如下：$(1,2,1,2,3,1,3)$，第一个数字1表示工件 $J_1$ 在第一台机器 $M_1$ 上的第一个位置，

第二个数字2表示工件$J_2$在第二台机器$M_2$上的第一个位置，第三个数字1表示工件$J_3$在第一台机器$M_1$上的第二个位置，第四个数字2表示工件$J_4$在第二台机器$M_2$上的第二个位置加工，其他定义类似。则可知工件在机器$M_1$上的顺序为$(J_1,J_3,J_6)$，机器$M_2$上加工顺序为$(J_2,J_4)$，机器$M_3$上的顺序为$(J_5,J_7)$。

由此，编码表的定义已经说明完成，任意解根据编码都可以准确地刻画出其具体排序方式，方便问题求解。

### 3.2 初始种群的设计

遗传算法的初始种群采取随机生成原则。根据本文问题和编码的特点，采取如下产生初始种群方法。

对于$n$个工件的集合$N$，按照工件的顺序，对每个工件，在满足资源约束的条件下任意选取一台机器进行加工，本文设置初始种群为15 个可行解。可行解产生的方法如下所述。

随机产生初始解，判断其是否满足资源约束条件，若满足，留下该可行解，不满足，舍弃。直到产生15 个可行解，记为初始种群$A=(X^1,X^2,\cdots,X^{15})$。

### 3.3 交叉变异以及选择策略的设计

交叉与变异运算是遗传算法的核心，它决定了算法的收敛速度以及优化结果。本文用目标函数值来衡量个体的适应度，针对本文的问题，目标函数为最小化最大完工时间$\min C_{\max}$，即：目标函数越小，适应度越高。同时对于交叉变异个体的选取方式如下所述。

(1)交叉概率与变异概率。对初始种群中的15 个可行解$A=(X^1,X^2,\cdots,X^{15})(k=1,2,\cdots,15)$，每个可行解随机产生交叉概率和变异概率，其中产生概率均为$[0,1]$之间的小数。为优化算法运行速度，本文设置交叉概率$P_c=0.4$，变异概率$P_m=0.1$。即，对某一可行解$X^k$，若其随机产生的交叉概率$P_{c_k}<0.4$，则$X^k$可用于进行交叉运算。若随机产生的变异概率$P_{m_k}<0.1$，则$X^k$可用于进行变异运算。该方法中，各个可行解被选中的概率只与其排列序号对应的概率值有关，而与其目标函数的具体数值无直接关系。

(2)交叉过程。针对本文工件的特性以及编码的设计思路，引入随机产生单点交叉的方式，减少对父代的依赖，增加种群的多样性。具体交叉方式如下：设$X=(x_1,x_2,\cdots,x_n)$$(j=1,2,\cdots,n)$为满足概率用于交叉的可行解，则从种群中剩下的个体中，随机选取可行解$X^k=(x_1^k,x_2^k,\cdots,x_n^k)$，随机生成交叉点$j$，只对两个个体交叉点之后的$(x_j,\cdots,x_n)$部分与$(x_j^k,\cdots,x_n^k)$部分进行交换，产生新的解$X'=(x_1,\cdots,x_j^k,\cdots,x_n^k)$和$X'^k=(x_1^k,\cdots,x_j,\cdots,x_n)$，作为接下来用于迭代的父代。例如：设有两条编码串为$X=(1,3,2,2,1)$，$Y=(2,1,3,1,2)$，若选取节点$j=3$进行交叉，则交叉后产生的解为$X'=(1,3,3,1,2)$，$Y'=(2,1,2,2,1)$。可以看到，从第三个工件开始，之后的工件排序方式就产生了互换。

(3)变异过程。变异过程是小概率发生的，能够对染色体产生较小的扰动来增加种群的多样性，针对本文问题的特点，采用单点变异方式，对于交叉后的染色体，进行简单的变异操作，该过程可理解为，让变异点处对应的工件，重新选取机器进行加工。设$X$为满足变异性用于变异的个体，其中$X=(x_1,x_2,\cdots,x_j,\cdots,x_n)$，$j=1,2,\cdots,n$，随机选取节点$j$进行变异，产生新的解$X'=(x_1,x_2,\cdots,x_j',\cdots,x_n)$，其中$x_j\neq x_j'$。即在本文中，对于工件$J_j$，随机重新选取机器加工，产生新的排序。例如，设有编码串为$X=(1,3,2,2,1)$，若选取节点$j=3$进行变异，随机产生新的解为$X'=(1,3,3,2,1)$，可以看到，只有第三个工件的机器选择发生了变化，但是它会对其他工件在机器上的顺序和整体的可行性产生影响。

(4)遗传算子的选取。针对本文特点，交叉变异后的个体，并不一定能够满足本文的资源约束条件，则对交叉变异后的新个体，进行可行性判定，若满足可行性判定，则令其$fitness=C_{\max}$，若不满足，则令其适应度的较大数值$fitness=10^{10}$。由此来筛选下一代的个体，同时为了避免原种群较优的个体丢失，融合新旧种群，对种群中所有个体的适应度由小到大排序，选取适应度最优的前15 个个体作为下一次迭代的父代。

### 3.4 算法的具体设计

步骤1：参数初始化。设定种群规模$Popnum=15$,最大迭代次数$Maxgen$，交叉概率$P_c$和变异概率$P_m$。

步骤2：生成包含15 个初始解的初始种群$chrom$,记为$(X^1,X^2,\cdots,X^{15})$。

步骤3：对于初始种群$(X^1,X^2,\cdots,X^{15})$,按照顺序选取当前解$X^q$，$q=1,2,\cdots,15$，随机产生的交叉概率$P_{c_k}$，判断是否满足$P_{c_k}<0.4$。若满足，则从种群中剩下的解中随机选取用于交叉的解$X^k$，进行交叉运算，产生新的个体,进行步骤4；若无需交叉，直接进行步骤4。

步骤4：同时对于步骤3产生的新个体，随机产生的变异概率$P_{m_k}$，判断其是否满足$P_{m_k}<0.1$，若满足，则继续随机选取节点的方式进行单点变异，产生新的个体，加入种群$newchrom$；若无需交叉，直接加入新种群$newchrom$。

步骤5：对于新旧种群进行合并产生$chrom^*=(chrom, newchrom)$，按适应度$fitness$由小到大排序，选取适应度排在前面的15 个个体，作为下一次迭代的种群记为$chrom$。同时输出融合种群中$C_{\max}$最小的解作为当前最优解，记为$X^*$。转到步骤2。

步骤6：当迭代次数满足设定次数数时，迭代结束，输出当前完工时间最小的解，记为$X^*=(x_1,x_2,\cdots,x_n)$，即可得到唯一的排序。

## 4 数值模拟及结果分析(Numerical simulation and result analysis)

采用数值计算的方式来验证所设计的遗传算法，算法采用了Python 3.10.1来实现算法和对数值实验的模拟，电脑的运行环境为Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHz 1.80 GHz和8GB Ram。本文的数值模拟实例通过随机产生，分别对小规模实例和大规模实例进行分析，目的是更好地证明该算法的有效性。

为了说明遗传算法对小规模实例的效果，本文设置机器数$m$=3，资源种类数为6，分别对工件数$n$为15、20、30，且工件大小分别从[0,10]、[0,30]、[0,50]中随机取数的不同类型进行了分析，每种情况随机产生20个实例，迭代次数为20。对每种类型的实例按以下方法求平均值进行对比，令每组产生的实例$\pi\in S$。

记遗传算法得到的算法解为$C_{ga}(\pi)$，记实例$\pi$计算得到的最优解下界为$C_B(\pi)$，最优解即为$C^*_{\max}$。定义$\alpha(\pi)=C_{ga}(\pi)/C_B(\pi)$，来表示本文遗传算法是算法解与最优解下界的比值。且定义实例的平均比值为$\alpha_{\text{avg}}=\sum_{\pi\in S}\alpha(\pi)/N$，$N$为该组实例中实例的个数。同时令$\alpha_{\min}$和$\alpha_{\max}$为该组实例最好、最坏情况下的$\alpha(\pi)$。通过小规模实例的结果分析可知，本文的算法在小规模体现出了较好的效果，可以看到，随着工件个数和工件大小的增加，其$\alpha_{\text{avg}}$稳定在1.2以内，是一个很接近最优解下界的比值。如表1所示。

表1 小规模实例运行20次的$\alpha_{\text{avg}}$数据结果

Tab.1 $\alpha_{\text{avg}}$ data results of a small-scale instance running for 20 times

| 工件数 | $\alpha_{\text{avg}}$ | | |
|---|---|---|---|
| | $p_j\in[0,10]$ | $p_j\in[0,30]$ | $p_j\in[0,50]$ |
| 15 | 1.094 | 1.117 | 1.139 |
| 20 | 1.132 | 1.081 | 1.115 |
| 30 | 1.172 | 1.154 | 1.166 |

同样需要验证算法对大规模实例的效果。增加工件数为100，资源种类数为20，迭代次数为50进行分析。工件大小分别从[0,30]、[0,50]、[0,100]中随机取数的不同类型进行分析，每类问题中工件的大小随机产生20个实例进行实验，对$\alpha_{\min}$、$\alpha_{\max}$和$\alpha_{\text{avg}}$进行对比，可以看到，当工件个数和大小持续增加时，$\alpha_{\text{avg}}$稳定在1.05以内，而最坏情况，也在1.2范围内，该遗传算法对于求解本文问题有着较好的优化作用。如表2所示。

表2 100个工件的实验结果

Tab.2 Experimental results of 100 workpieces

| 工件大小 | $\alpha_{\min}$ | $\alpha_{\max}$ | $\alpha_{\text{avg}}$ |
|---|---|---|---|
| $p_j\in[0,30]$ | 1.007 | 1.090 | 1.036 |
| $p_j\in[0,50]$ | 1.005 | 1.104 | 1.039 |
| $p_j\in[0,100]$ | 1.005 | 1.134 | 1.047 |

现给出随机产生100个工件的实例，工件大小从[0,100]中随机取数，利用遗传算法迭代100次的迭代图。通过Python软件matplotlib.pyplot来实现，迭代结果如图1所示，在该实例中，遗传算法解为$C_{ga}(\pi)$=1,733，最优解下界为$C_B(\pi)$=1,672，算法解与最优解下界比值$\alpha(\pi)$=1.036，算法运行时间为67.76 s，是一个较为理想的结果，其收敛速度也较快，虽然在可行性的判定过程中增加了运行时间，但是在可接受的范围内。进一步说明了遗传算法对本文问题具有较高的优化作用。

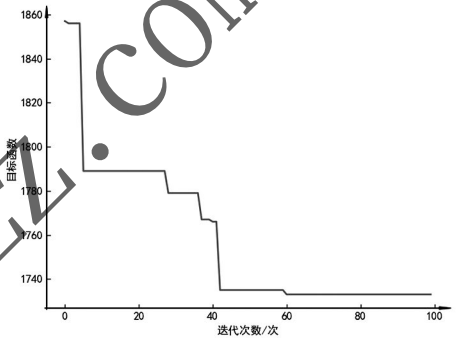


图1 遗传算法迭代收敛图

Fig.1 Iterative convergence diagram of genetic algorithm

综上，本文设计的遗传算法针对问题$P_m|res\lambda 11|C_{\max}$有着较好的优化效果，是针对该问题的一个有效算法。

## 5 结论(Conclusion)

遗传算法在组合优化和车间调度领域有着广泛的应用，对于资源约束的平行机排序，有着较好的优化效果。首先，针对本文问题$P_m|res\lambda 11|C_{\max}$，设计遗传算法在初始种群的产生时，就加入了可行性判定，避免对无效的解进行迭代，提高了算法的有效性。其次，在算子的选择和交叉变异过程中，基于概率原则，而不是确定的解，增加了解的多样性，利于全局搜索。同时针对本文问题，以目标函数值最小化最大完工时间来作为算法的适应度，更有利于解的优化。最后，本文通过随机产生实例进行了数值实验，在随机产生的大量实例中，算法解和最优解下界的比值稳定在1.2以内，证明了遗传算法对于本文问题的有效性,对于资源约束的平行机调度问题有着较好的优化效果。

## 参考文献(References)

[1] KELLERER H, STRUSEVICH V A. Scheduling parallel dedicated machines under a single non-shared resource[J]. European Journal of Operational Research, 2003, 147(2): 345-364.

(下转第49页)