基于启发式方法的车间作业排序问题的三步优化

林理露

2016 年 6 月 8 日

摘要: 启发式规则实现简单, 运行效率高, 但是求解效果差, 且有对问题形式的较强的依赖性, 本文采用了 JMP 启发式规则，引入多个优先级因素，充分考虑了机器、工件、工序间的负载均衡。元启发式算法概念简明，实现方便，在解决复杂组合优化类问题方面具有优越性能，但容易早熟，而陷入局部最优状态，全局寻优能力有限, 本文采用了 PSO 粒子群算法，并针对早熟问题，引入权重自适应、随机性限定解空间范围、使用 ROV 规则解码等技术，提高了 PSO 算法的全局寻优能力，使其具有了较快的收敛速度。超启发式算法通过上层的元启发框架来控制底层的多个启发式规则，兼具了启发式规则的运算高效性和元启发式算法的全局寻优能力，是现今复杂优化问题的最前沿、最优秀的技术，本文使用 GA 算法作为元启发框架，另使用 6 个基础的启发式规则，实现了最佳的问题求解水平。通过单一启发式规则到元启发式算法到超启发式算法的三步优化，有效地解决了最基本的车间调度问题中的调度规划问题，同时也为柔性作业车间调度问题、多工艺路线车间调度问题、多时间因素作业车间调度问题提供了思路。

关键词: 车间作业排序 启发式规则 粒子群算法 超启发式算法 遗传算法

1 引言

车间调度问题 (Job Shop Scheduling Poblem, JSP) 是一个多约束组合优化和非多项式确定 (Non-Deterministic Polynomial, NP) 难题。求解 JSP 问题主要有精确算法和启发式优化方法 (近似算法) 两种方向。前者主要有解析、整数规划、分支界限法等, 由于时间复杂度较高，只能用于求解一些小规模的问题。而实际生产中的调度问题, 由于其问题规模大, 复杂度高, 且因素多，精确调度算法很难得到应用，故主要研究方向是各种近似算法。早期出现的最基础的启发式规则求解效率高，在很多情况下可以快速地得到很不错的计算结果，但是按照启发式规则所得到的调度的精度在大多数的情况下还不能满足实际生产的要求。近年来，模拟退火算法、禁忌搜索、遗传算法、蚁群算法、粒子群算法、免疫算法等元启发式算法相应被用于求解 JSP 问题, 这类算法优化能力强，取得了一定的优化的效果，基本上能满足现代制造业企业的要求。但是由于车间作业调度优化问题的复杂性, 以及启发式算法存在的各种不足, 至今尚未形成系统的理论与方法。而最新出现的超启发式算法基于前两者，使用高层的元启发式算法框架来控制底层多个启发式规则, 结合了前两者的优势, 兼具了启发式规则的运算高效性和元启发式算法的全局寻优能力，是现今复杂优化问题的最前沿、最优秀的技术。本文分别使用以上三种优化方法来逐步寻求对基础 JSP 问题的优化，并从中展现出启发式方法的进化历程及最新的启发式方法的优越性。本文中，启发式规则使用了 JMP 算法 (考虑繁忙度的前沿贪心方法)，元启发式算法使用了 PSO 算法 (粒子群算法), 超启发式算法使用了基于 GA(遗传算法) 的框架，控制底层 FA、EFT、LU、MA、SPT、TIS 六个启发式规则。并基于 C++ 对以上三种启发式方法分别进行性能测试。

2 问题描述与模型的建立

2.1 问题描述

一个最经典的车间作业排序问题可描述为: 一个加工系统中有 m 台机器和 n 个待加工的工件, 所有工件的加工路径 (即机器约束) 预先给定, 但不要求一致, 各

工件在各机器上的操作时间已知。排序的任务是通过一定的优化，合理地安排每台机器上工件的加工次序，使约束条件得到满足，同时使车间作业的总调度时间最短。

最典型的 Job Shop 问题一般会有以下约束条件:

1. 同一时刻每台机器只能加工一个工序，且每个工序只能被一台机器所加工，同时加工过程不可间断;
2. 在整个加工过程中，每个工件不能在同一台机器上加工多次;
3. 各工件必须按照工艺路线以指定的次序在机器上加工，工件 i 的第 j 道工序必须在第 (j -1) 道工序完成后才能开始;
4. 不考虑工件的优先权，允许操作等待;
5. 工件的加工时间事先给定，且在整个加工过程中保持不变。

目标函数是最小化最大完工时间，即

$$T = min\{\underset{1\le j\le m}{Max}(c_j)\}$$

其中 c_j 表示第 j 号机器的完工时间

2.2 数学模型

2.2.1 已知条件

首先要建立工序排列矩阵 J，各工件加工时间矩阵 T，各机器加工工序的调度矩阵 M_J

1. 需要加工的工件集为: $J = (j_1, j_2, j_3, \ldots, j_n)$,$j_i$ 为第 i 个要加工的工件;
2. 能够加工的机器集为: $M = (m_1, m_2, m_3, \ldots, m_m)$,$m_i$ 为第 i 个加工用的机器 ;
3. 各工件的工序集为: $S = (S_1, S_2, S_3, \ldots, S_n)^T$;$J_i = (j_{i1}, j_{i2}, j_{i3}, \ldots, j_{i_m})$;$j_{ik}$ 表示第 i 个工件的第 k 道工序;
4. 工件各工序的加工时间 $T = (T_1, T_2, T_3, \ldots, T_n)^T$;$T_i = (t_{i1}, t_{i2}, t_{i3}, \ldots, t_{i_m})$;$t_{ik}$ 表示第 i 个工件的第 k 道工序的加工时间;

2.2.2 约束条件

$$\begin{cases} St_{ij} + t_{ij} \le St_{i(j+1)}, \\ St_{ijl} + t_{ij} \le St_{ghk}, \\ t_{ij} \ge 0, \\ i, g, k = 1, 2, 3, \ldots, n, \\ h, j = 1, 2, 3, \ldots, m, \end{cases}$$

式中，St_{ij} 代表工件 i 的第 j 道工序的开工时间

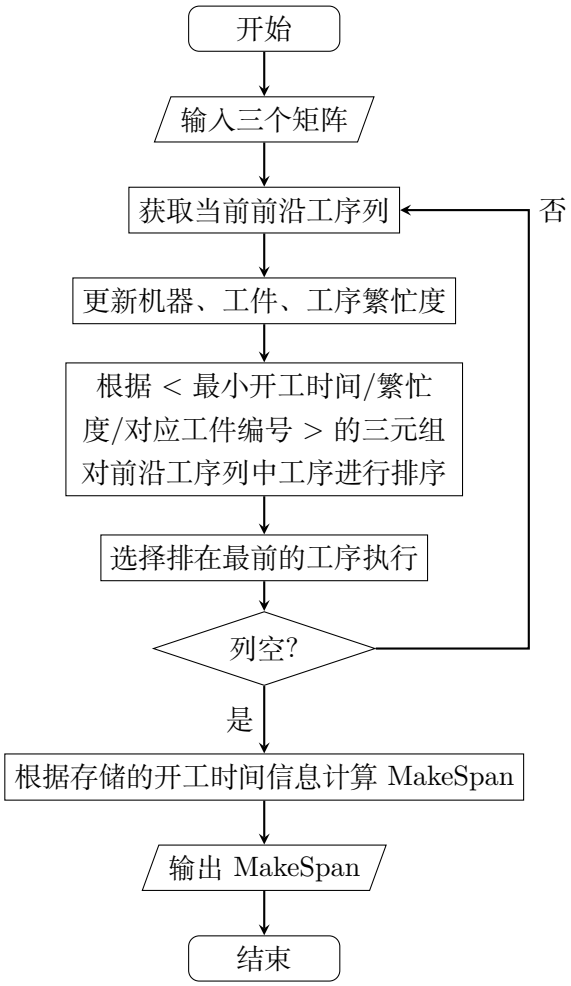
2.2.3 目标函数

车间作业调度问题优化的目标函数为:

$$Min(T(J_m)) = min\{max[T_1, T_2, T_3, \ldots, T_m]\}$$

3 JMP 启发式规则

3.1 流程图



3.2 启发式规则的车间作业调度求解

3.2.1 名词定义

前沿工序 指所有未被调度的工序当中，在所在工件中所排最靠前的工序集合

繁忙度 通过特定的规律计算出的反映繁忙程度的量

工件繁忙度 JobBusy(i)=(属于工件 i 的未处理的工序数 × 未处理的工序所需时间和)

机器繁忙度 MachineBusy(k)=(属于机器 k 的的未处理的工序数 × 未处理的工序所需时间和)

工序繁忙度 Busy(i)= 工件繁忙度 + 机器繁忙度

最小开工时间 即在当前已安排的工序的格局之下，该工序能开始执行的最早的时间点

3.2.2 算法流程

该基于繁忙度的计算过程被命名为 JMP 启发式规则，步骤如下：

1. 在最开始的格局之下，所有工序都还没有执行
2. 选择出当前的前沿工序
3. 利用 < 最小开工时间/工件繁忙度/工件所在编号 > 的三元组，对前沿工序进行排序，最小开工时间小的、工件繁忙度高的、工件序号小的优先。由约束条件可知，两个前沿工序工件序号不可能相同，故该排序肯定能产生一个有序序列
4. 选取排序后序列中的最佳工序执行
5. 按照如上规则执行完所有工序，并得到一个合法的相应的加工周期 MakeSpan

3.3 优缺点分析

优点

- 时间复杂度低，消耗计算资源较少
- 逻辑简单，易于实现
- 代码量低，嵌入硬件难度小

缺点

- 过分依赖具体问题类型，不具备通用性
- 很难保证解的质量，可能产生很差的解
- 问题条件发生变化时，需完全重构，适应性差

4 PSO 粒子群算法

PSO 粒子群算法，是一种模拟鸟群体飞行的群智能算法，与其它的优化算法基本思想相似，在 PSO 中，一个粒子表示一只鸟，每一个粒子均具有初始位置 X 和速度 V，在粒子群飞行过程不断调整飞行速度和方向，最终找到最优解，它用无质量、无体积的粒子作为个体，并为每个粒子规定简单的行为规则，从而使整个粒子群表现出复杂的特性，可用来求解复杂的优化问题。

4.1 数学描述

设在一个 n 维的搜索空间中，由 m 个粒子组成的种群 $X = \{x_1, \ldots, x_i, \ldots, x_m\}$，其中第 i 个粒子位置 $x = (x_{i1}, x_{i2}, \ldots, x_{in})^T$，其速度 $V_i = v_{i1}, v_{i2}, \ldots, v_{in})^T$，个体极值 $Pb_i = (P_{i1}, P_{i2}, \ldots, P_{in})$，种群的全局极值 $P_g = min(P_{i1}, P_{i2}, \ldots, P_{in})$

设粒子 i 的速度和位置分别定义为 $X_i(t)$ 和 $V_i(t)$，在每一次迭代过程中，粒子跟随自身最优解 (P_{best}) 和种群最优解 (G_{best}) 来更新自己的速度和位置，具体的更新公式如下:

$$V_{ik}(t+1) = \omega V_{ik} + c_1 r_1 (P_{ik}(t) - X_{ik}(t)) + c_2 r_2 (P_{gk}(t) - X_{ik}(t))$$

$$X_{ik}(t+1) = Xik(t) + V_{ik}(t+1)$$

式中，ω 表示粒子的惯性权重,c_1, c_2 表示加速因子数,r_1, r_2 为分布于 [0,1] 之间的随机数，t 为当前进化代数,$k = 1, 2, \ldots, n; i = 1, 2, \ldots, m$ 为种群规模。

4.2 参数设置

4.2.1 最大速度 $Vmax$

在上述粒子速度、位置更新公式中，速度 V_{ik} 和 X_{ik} 的绝对值可能会过大,从而使粒子一下子飞出解空间。因此要将其限制在 $[-V_{max}, V_{max}]$ 和 $[-X_{max}, X_{max}]$ 内,本文将 V_{max} 设为 1,X_{max} 设为 5，同时引入了随机化的

因素，当粒子脱离限制的范围时，按如下方法规范粒子速度和位置：

$$\pm X_{ij} = \pm X_{max} \times (1-r), r \in [0, 0.1]$$

$$\pm V_{ij} = \pm V_{max} \times (1-r), r \in [0, 0.1]$$

4.2.2 权重因子 ω

惯性权重 ω 使粒子保持运动惯性，使其有扩展搜索空间的趋势，文献 [2] 表明，当 $\omega \in [0.9, 1.2]$ 时，粒子群算法能获得较好的优化效果，而当 ω 从 0.9 递减至 0.4 时，算法能很快地向最优解收敛，故本文使用自适应的惯性权重系数，其在 PSO 的搜索中线性变化，计算公式如下:

$$\omega = \omega_{min} + \frac{\omega_{max} - \omega_{min}}{n} \times i$$

4.2.3 加速度常数 c_1, c_2

加速度常数 c1 和 c2 代表将每个粒子推向 P_{best} 和 G_{best} 位置的统计加速项的权重。较低的值允许粒子在被拉回之前可以在目标区域外徘徊，而较高的值则导致粒子突然越过目标区域。根据文献 [2]，本文将 c_1, c_2 取值为 2。

4.3 编码

车间作业调度问题是一个离散、动态、多变量的问题，而粒子群优化算法是一种连续空间的优化算法，所以在利用 PSO 求解车间作业调度问题时，粒子位置不能直接用于表示最终调度的序列，二者之间需要建立一种映射关系。由于寻优目标为排序问题，每一个粒子代表的是工件的一个排序，因此粒子的运动代表的是工序的改变，如此便很难保证工序的合理性，为此，本文采用了基于操作的编码方式，使用 ROV 规则来实现了粒子位置到工序操作的映射。

该编码方式方式使用 m×n 个代表操作位置值表示微粒的位置矢量，即所有操作的一个序列，其中每个工件号均出现 m 次。解码过程是：先将微粒位置转化为一个有序的操作表，然后基于操作表和工艺约束将操作表中的元素按照从左到右的顺序解释为相应工件的操作顺序，进而产生调度方案。

基于操作的编码方式的基本思想是将所有工件的操作进行编码，不同工件用不同的编码表示，同一工件则用相同的编码表示。编码步骤如下：

ROV 规则编码

1. 随机初始化微粒 $X_k = [x_{1,1}, x_{1,2}, \ldots, x_{1,n}, x_{2,1}, x_{2,2}, \ldots, x_{2,n}, \ldots, x_{m,1}, x_{m,2}, \ldots, x_{m,n}]$ 其中 n 表示工件数,m 表示机器数量。

2. 分别对 X_k 中的子部分 $[x_{i,1}, x_{i,2}, \ldots, x_{i,n}], i = (1, 2, \ldots, m)$ 使用 ROV 规则。ROV 规则为: 比较数组中元素值的大小，按从小到大依次将其标注为 1,2,...,n。

3. 将上述微粒位置经过 ROV 规则转换后，将微粒位置矢量映射为相应的整数，而整数则可用来代表相应的工件序号。操作则可用其对应的工件号来表示，操作可用符号 O_{ijk} 表示，表示第 i 个工件的第 j 道工序在第 k 台机器上加工。

如,n=3,m=2 时，设微粒初始化位置为:X_k=[1.32 1.16 2.10 2.13 1.83 1.56] ROV 变换之后，则 X_k=[2 1 3 3 2 1]; 工艺约束和加工时间如下表所示，则该微粒位置对应的工件加工顺序为 $[O_{212}O_{111}O_{312}O_{321}O_{221}O_{122}]$。

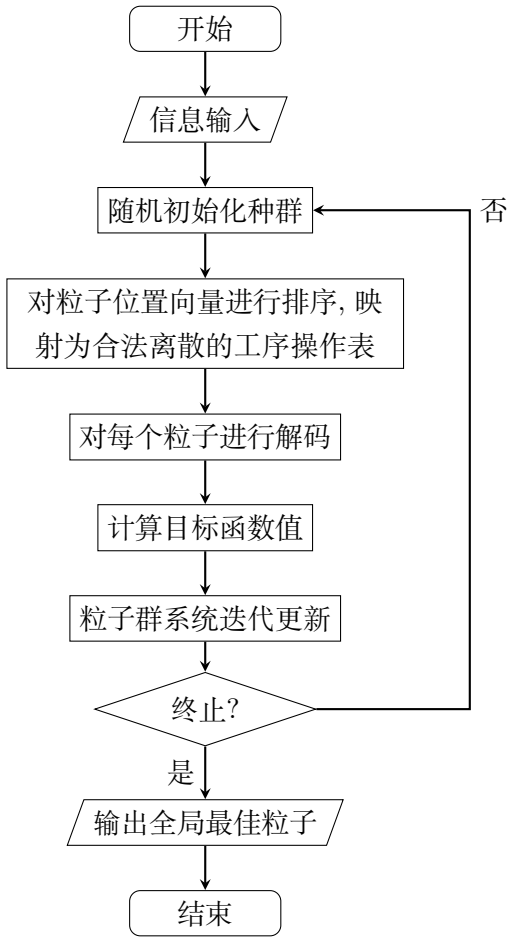
加工时间和工艺约束表

		操作序列	
项目	工件	1	2
操作时间	J_1	3	3
	J_2	1	5
	J_3	3	2
机器	J_1	M_1	M_2
	J_2	M_2	M_1
	J_3	M_2	M_1

4.4 目标函数和适应值的计算

已知每个工件每道工序在机器上的加工时间段 t_ij，向量 $T_m = [t_{m1}, t_{m2}, \ldots, t_{mm}]$，其中，$t_{mj}$ 代表工件在第 j 台机器上的累计加工时间，向量 $T_P = [t_{P1}, t_{P2}, \ldots, t_{Pn}]$，其中，$t_{Pi}$ 代表第 i 个工件的累计加工时间·初始化向量 T_m, T_P 各分量都为 0·按编码所确定的顺序将工件分配到相应的机器上，在分配时同时考虑机器约束和工件约束，将加工时间 t_{ij} 分别对应地加到向量 T_m 和向量 T_P 上·按照上述方法得到向量 T_m 中最大的分量值即为加工完成时间。

4.5 流程图



4.6 PSO 的车间作业调度求解

主要算法流程如上图所示，部分详细操作如下:

4.6.1 粒子更新操作

将每个粒子适应度 (本文直接使用 MakeSpan) 值与 P_{best} 和 G_{best} 比较，若优于 P_{best} 或 G_{best}，就采用该粒子适应度值代替 P_{best} 或 G_{best}。

4.6.2 终止条件判断

当已达到最大迭代数或者连续 K 代粒子群全局最优值不发生大的变化，则表示满足终止条件，输出最优解。

4.7 优缺点分析

优点

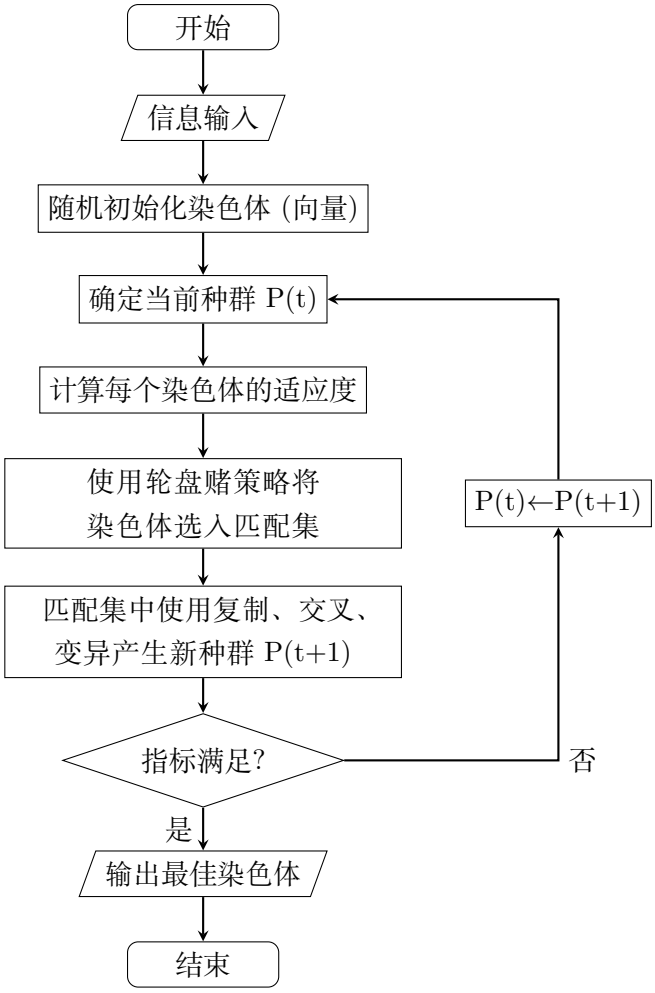
- 个体数目少，相比 ACO 等算法，PSO 个体数目较少，便于操作
- 计算简单，粒子更新公式单一，逻辑相对简单
- 健壮性强，对初始解的依赖性小，即使是随机生成的初值也总能找到较优的解

缺点

- 容易收敛过早，陷入局部最优解
- 代码量大，且运算复杂，不便于嵌入硬件
- 编码方式导致搜索空间为规则集, 部分特殊粒子难以被选中
- 随机策略较少，多次求解结果相似度高

5 基于 GA 的 HHA 超启发算法

5.1 流程图



5.2 HHA 的车间作业调度求解

超启发式算法作为最新、最有前景的组合优化式方法,采用高层的元启发式方法搜索低层的启发式规则,使得算法兼备寻优能力与计算效率。本文使用了 GA 的高层框架，并使用了 6 个底层启发式规则。

5.2.1 GA 框架

染色体 染色体为 GA 算法的基础，全部遗传算子都是建立在染色体上的，编码的优劣决定了遗传算法的质量。而对于传统的元启发式遗传算法，遗传算子直接代表着工序，而工序间有诸多约束，于是仍然需要添加额外的检测和克服不可行调度的算法, 增加了系统不必要的开销。同时在执行交叉和突变的操作上仍需避免产生不可行调度。而基于 GA 的超启发式方法中，染色体对应着某一具体的启发式规则，而启发式规则间并无任何约束，因此可以自然的全空间的编码，而不用担心产生不可行调度的问题。本文选取 $D_t = (d_1, d_2, \ldots, d_m)$ 作为染色体，其中 $d_k = (1, 2, \ldots, 6), k = (1, 2, \ldots, m)$ 代表着对应编号的启发式规则,t 代表当前染色体代数。

匹配集 代表着能适应当前环境，能够遗传产生后代的候选染色体集，相当于一个缓冲区，为以后染色体交换、变异，产生新的染色体作准备。

选择 即从原有的染色体集中直接选择一条复制至匹配集中，即 $D_{t+1} \leftarrow D_t$。

交叉 选择操作不能创新, 交叉操作可以解决染色体的创新,交叉即为随机选择两个染色体 (双亲染色体), 随机指定一点或多点, 进行交换, 可得两个新的染色体 (子代染色体), 本文采用随机选择起始点和终止点，交换中间所有部分的方式，模拟了自然界的染色体交叉现象，即 $D_k = (d_{i1}, d_{i2}, \ldots, d_{js}, d_{js+1}, \ldots, d_{jt}, d_{it+1}, \ldots, d_{im}) \leftarrow (Di, Dj)$ $s, t = 1, 2, \ldots, m$ 分别代表交叉开始和结束的位置。本文根据文献 [7] 将交叉概率设为 P_c0.65。

突变 突变模拟生物在自然界环境变化，引起基因的突变。在染色体编码中，由原来的基因突变为其他基因。即 $D_i = (d_{i1}, d_{i2}, \ldots, d_k, d \ldots, d_m)$ 其中 $d_k = (1, 2, 3, \ldots, 6)$ 和其他基因是否发生变化相互独立。突变产生染色体的多样性，避免进化中早期成熟，陷入局部极值点。突变的概率很低，本文根据经验设置突变概率 P_m=0.005。

解码方式 以染色体中编码对应编号调用相应的启发式规则，即可求得对应 MakeSpan。

适应度 适应度计算，根据文献 [9]，本文采用如下公式计算每个染色体的适应度:

$$fit(x) = \frac{1}{Obj(x) + 1}$$

其中 x 代表当前染色体，Obj(x) 代表染色体解码后的 Makespan 长度。

环境变量 由于直接使用轮盘赌，染色体被选入的概率过小，往往还未进行迭代就全部灭绝，导致 GA 框架无法运行, 故本文引入了环境变量 EV 来提高染色体被选入匹配集的概率。而单一环境变量会导致算法寻优能力受限，故本文使用动态环境变量，更新公式如下：

$$EV_{t+1} = EV_t \times \frac{t_m - \frac{2}{3}t}{t_m}$$

其中 t 为当前染色体遗传代数, t_m 为最高代数, 整个环境在逐渐变差，从而更高效地淘汰掉不能很好地适应环境的染色体，使算法结果更快收敛向最优解。

轮盘赌策略 采用适应度比例法（轮盘赌）按各染色体适应度大小比例来决定其被选择数目的多少。轮盘赌决定概率，引入环境变量调整宏观平均生存率。公式如下:

$$P_s = EV \times \frac{f(x_i)}{\sum f(x_i)}$$

其中 EV 是环境变量，f 为适应度函数，x_i 代表特定的一条染色体,P_s 代表被选入匹配集的概率。

5.2.2 启发式规则集

启发式规则表

编号	规则名	描述	公式
1	FA	先到先得	$min(Tp_d)$
2	EFT	最早完工时间	$min(Max(Tp_s))$
3	LU	最低使用率	$min(\frac{Tp_d}{Tp_s})$
4	MA	最高空闲率	$max(\frac{Tp_s - T_s - Tp_d}{Tp_s})$
5	SPT	最短加工时间	$min(T_{s_i})$
6	TIS	机器中时间最长	$min(Tp_s - Tp_d)$

其中，Tp_d 指当前格局下，该工序对应工件已完成的工序所用时间，Tp_s 为完成整个工件所需总时间，T_s 为加工该工序所需时间。

5.3 优缺点分析

优点

- 结合了元启发式算法的寻优能力和启发式规则的计算效率
- 可以将底层启发式规则嵌入硬件，用高层程序控制底层启发式规则
- 灵活性高，对问题适应性强，条件改变时算法修改较少
- 编码对应启发式规则，基本无约束，实现方便

缺点

- 逻辑复杂，实现难度高
- 由于本文启发式规则量少，难以搜索整个解空间

6 算法仿真

6.1 实验环境

硬件仿真环境:Intel i5-5257U 2.7-2.9GHz/8GB 1867MHz DDR3

软件平台:OSX 10.11.5 系统，编译器 Apple LLVM version 7.3.0 (clang-703.0.29)

6.2 结果

为了验证三种启发式算法求解 JSP 的性能，考虑采用一个典型 JSP 算例 (LA 类)，这些测试问题在文献中被广泛用于测试。FT 类问题由 Fisher and Thompson 给出，LA 类问题由 Lawrence 给出，在此选取其中 6 个不同规模的问题:FT06，FT10，LA01，LA05，LA10，LA12，各算法均独立运行 25 次。其中迭代次数上限均为 600，种群规模均为 50，其余参数按前文所述设置，实验结果如下表所示:

问题	算法	最优解	平均解	最差解	平均用时
$FT06_{6\times6}$	JMP	77	77	77	0.171ms
	PSO	59	59	59	0.193s
	HHA	65	68	74	0.905s
$FT10_{10\times10}$	JMP	1260	1260	1260	0.171ms
	PSO	1212	1220	1231	0.598s
	HHA	1254	1363	1444	4.712s
$LA01_{5\times5}$	JMP	822	822	822	0.171ms
	PSO	724	724	725	0.303s
	HHA	755	797	933	1.249s
$LA05_{10\times5}$	JMP	759	759	759	0.362ms
	PSO	593	593	594	0.311s
	HHA	624	657	712	1.007s
$LA10_{10\times5}$	JMP	1189	1189	1189	0.537ms
	PSO	958	958	959	0.485s
	HHA	973	1025	1105	1.688s
$LA12_{20\times5}$	JMP	1240	1240	1240	0.612ms
	PSO	1039	1039	1040	0.695s
	HHA	1039	1134	1315	2.225s

6.3 三种算法的对比分析

从启发式规则到元启发式算法再到超启发式算法，代码逻辑逐渐变得复杂、构建难度逐步提升，有必要对三者进行比较。显然单纯的启发式在速度上有绝对优势，比其他两种算法快了 3 个数量级。而 PSO 粒子群算法很好的兼顾了性能与结果，而对于的超启发式算法，由于代码逻辑复杂，运行时间较长，且本文中仅有 6 个启发式规则，启发式规则集过小，因此寻优能力一般，但显然强于单纯的启发式规则，在大规模问题下寻优能力稍有增强。

7 结论

通过基于启发式方法的对车间作业调度问题的三步优化，本文充分体现出了启发式方法在求解 NP 完全问题中的成长、进化过程。虽然本文中由于启发式规则集容量较小而导致了超启发式算法的性能与结果不够突出，但可以预见的未来之中，超启发式算法应该会是求解 NP 完全问题中的最有前景、最具发展潜力的算法，将会是今后研究的重点方向。

参考文献

[1] 曾立平, 黄文奇. 一种用于车间作业调度问题的智能枚举算法 *[J].* 计算机工程与应用. 2004(30).
Zeng Liping,Huang Wenqi. *An Intelligent Enumeration Algorithm for Job Scheduling Problem[J].* Computer Engineering and Applications. 2004(30)

[2] 何　利, 刘永贤, 谢华龙, 刘笑天. 基于粒子群算法的车间调度与优化 *[J].* 东北大学学报（自然科学版). 2008(04)
HE Li,LIU Yong-xian,XIE Hua-long,LIU Xiao-tian *Job Shop Scheduling and Its Optimization Based on Particle Swarm Optimizer[J].*Journal of Northeastern University(Natural Science). 2008(04)

[3] 黄慧, 黎向锋, 左敦稳, 薛善良. 基于改进粒子群算法的车间作业排序的优化设计 *[J].* 中国制造业信息化. 2011(21)
HUANG Hui,LI Xiang-feng,ZUO Dun-wen,XUE Shan-liang *Design of Job Shop Scheduling Based on Improved Particle Swarm Algorithm* Manufacture Information Engineering of China. 2011(21).

[4] 张飞, 耿红琴 基于混沌粒子群算法的车间作业调度优化 *[J].* 山东大学学报 (工学版). 2013(03)
ZHANG Fei,GENG Hong-qin. *Optimization of job shop scheduling problem based on chaos particle swarm optimization algorithm[J].* Journal of Shandong University(Engineering Science). 2013(03)

[5] 毛帆, 傅鹂, 蔡斌 求解作业车间调度问题的微粒群遗传退火算法 *[J]* 计算机工程与应用. 2011(05)
MAO Fan,FU Li,CAI Bin. *Particle swarm genetic annealing algorithm for job-shop scheduling.Computer Engineering and Applications[J].* 2011,47(5):227-231.

[6] YAN Ping, JIAO Minghai. *An Improved PSO Search Method for the Job Shop Scheduling Problem* 2011 Chinese Control and Decision Conference (CCDC)

[7] 谢胜利, 黄强, 董金祥. 求解 *JSP* 的遗传算法中不可行调度的方案 *[J].* 计算机集成制造系统-CIMS. 2002(11)
XIE Sheng-li 1,2 ,HUANG Qiang 2 ,DONG Jin-xiang 2 *A Method to Resolve Unfeasible Scheduling of JSP by GA[J].* Computer Integrated Manufacturing Systems, 2002(11)

[8] Ender Ožcan,Burak Bilgin and Emin Erkan Korkmaz. *A comprehensive analysis of hyper-heuristics* Intelligent Data Analysis 12 (2008) 3–23

[9] Dongni Li, Rongxin Zhan, Dan Zheng, Miao Li, and Ikou Kaku. *A Hybrid Evolutionary Hyper-Heuristic Approach for Intercell Scheduling Considering Transportation Capacity* IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, VOL. 13, NO. 2, APRIL 2016.

Written using LaTeX by $\mathcal{L}in\mathcal{L}i\mathcal{L}u$

Special Thanks to $\mathcal{D}ong\mathcal{N}i\ \mathcal{L}i, \mathcal{R}ong\mathcal{X}in\ \mathcal{Z}han$