

哈爾濱工業大學

計算建模  
大項目結題報告

題 目	<u>模擬退火算法解決流水車間調度問題</u>
學 院	<u>計算機科學與技術</u>
專 業	<u>計算機科學與技術</u>
學 號	<u>1190200122</u>
學 生	<u>袁 野</u>
任 課 教 師	<u>范曉鵬 劉紹輝</u>

哈爾濱工業大學計算機科學與技術學院

2021. 12

# 模拟退火算法解决流水车间调度问题

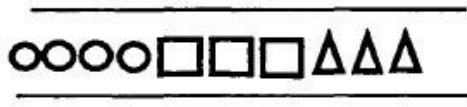
## 一、 项目内容梳理 (20 分)

如今，为了满足客户多样化与个性化的需求，多品种、小批量生产已经为一种重要的生产方式。与过去大批量、单一的生产方式相比，多品种、小批量生产可以快速响应市场，满足不同客户的不同需求，因此，受到越来越多的企业管理者的重视。特别是以流水线生产为主要作业方式的企业，企业管理者致力于研究如何使得生产均衡化，以实现生产批次的最小化，这样可以在不同批次生产不同品种的产品。在这种环境下，对于不同批次的产品生产进行合理调度排序就显得十分重要。

在传统的生产方式中，企业生产者总是力求通过增加批量来减小设备的转换次数，因此在生产不同种类的产品时，以产品的顺序逐次生产或用多条生产线同时生产。这样，必然会一次大批量生产同一产品，很容易造成库存的积压。在实际生产中如果需要生产 A, B, C, D 四种产品各 100 件，各种产品的节拍都是 1 分钟，如果按照传统的做法，先生产出 100 件 A 产品，其次是 B，然后是 C，最后生产产品 D。在这种情况下，这四种产品的总循环时间是 400 分钟。然而，假设客户要求的循环时间为 200 分钟(四种产品的需求量为 50 件)，那么在 200 分钟的时间内就只能生产出产品 A 和产品 B，因而不能满足客户需求，同时还会过量生产产品 A 和 B，造成库存积压的浪费。这种生产就是非均衡的，如图 1 所示。

比较均衡的生产方式(图 2 )是:在一条流水线上同时将四种产品混在一起生产，并且确定每种品种一次生产的批量。当然，如果在混合生产时不需要对设备进行转换，那么单件流的生产方式是最好的。然而，在实际生产 A, B, C, D 四种不同产品时，往往需要对流水线上的某些设备进行工装转换。单件流的生产方式在此难以实现，需要根据换装时间来确定每种产品一次生产的批量。同时，由于现实生产中不同产品在流水线上各台机器的加工时间很难相同，因此，流水线的瓶颈会随着产品组合的不同而发生变化。当同一流水线加工多产品，并且每种产品在各道工序(各台机器)的加工时间差异较大时，瓶颈就会在各道工序中发生变化，如何对各种产品的投产顺序进行优化以协调这些变化的瓶颈是生产管理

中一个很重要的问题。



非均衡的生产方式

图 1



均衡化的生产方式

图 2

因而对流水线调度问题的研究正是迎合这种多品种、小批量生产方式的需  
要，我们要讨论得是如何对流水线上生产的不同产品的调度顺序进行优要化。

流水车间调度问题一般可以描述为  $n$  个工件要在  $m$  台机器上加工，每个工  
件需要经过  $m$  道工序，每道工序要求不同的机器， $n$  个工件在  $m$  台机器上的加  
工顺序相同。工件在机器上的加工时间是给定的，设为

$$t_{ij}(i = 1, \dots, n; j = 1, \dots, m)$$

问题的目标是确定个工件在每台机器上的最优加工顺序，使最大流程时间  
达到最小。

对该问题常常作如下假设：

- (1) 每个工件在机器上的加工顺序是给定的；
- (2) 每台机器同时只能加工一个工件；
- (3) 一个工件不能同时在不同机器上加工；
- (4) 工序不能预定；
- (5) 工序的准备时间与顺序无关，且包含在加工时间中；
- (6) 工件在每台机器上的加工顺序不同，且分别是确定的。

问题的数学模型：

$c(j_i, k)$ ：工件  $j_i$  在机器  $k$  加工的完工时间。

$\{j_{i,1}, j_{i,2}, j_{i,3}, \dots, j_{i,n}\}$ ：第  $i$  台机器的工作调度。

$n$  个工件， $m$  台机器的流水车间调度问题的完工时间：

$$C(j_{1,1}, 1) = t_{j_{1,1},1}$$

$$C(j_{i,k}, i) \geq C(j_{i-1,k}, i-1) + t_{j_{i-1,k},i-1}$$

$$C(j_{i,k}, i) \geq C(j_{i,k-1}, i) + t_{j_{i,k-1}, i}$$

$$C_{\max} = C(j_{m,n}, m)$$

调度目标：确定 j 矩阵，最小化  $C_{\max}$

分工如下：

袁野	模拟退火和评价函数的代码编写，测试
陆任驰	此时数据构造，文献查阅
王一凡	流水车间调度问题其他方法学习总结

## 二、 特定主题(topic)或特定文献的内容介绍-发展现状和发展趋势， 所研究主体的国内外现状，最新进展等（20 分）

自从 Jhonson 1954 年发表第一篇关于流水车间调度问题的文章以来，流水车间调度问题引起了许多学者的关注，目前存在着以下几种解决方法：

### 1、一种基于扩展采样空间的混合式遗传算法

将邻域搜索与遗传算法相结合求解流水车间调度问题，提出了一种邻域结构，使之更适合求解流水车间问题；设计了一种基于扩展采样空间的混合式遗传并通过计算机模拟验证其有效性。其中，邻域搜索使用定义（由给定的染色体通过随机移动一个基因到一个随机的位置，得到的是染色体的集合）所描述的邻域，采样空间为父代  $P(t)$ 、改进的父代  $S(t)$ 、交叉的后代  $C(t)$ 、变异的后代  $M(t)$ ，交叉和变异的父代是种群的父代  $P(t)$ ，而不是改进的父代  $S(t)$ 。

具体的混合式算法框架

BEGIN

$t=0$

初始化  $P(t)$

WHILE 不满足终止条件 DO

① 下降搜索，应用多点最速下降法改进  $p(t)$ ，得到改进的父代  $S(t)$ ：

② 用  $P(t)$  进行单点交叉生成  $C(t)$ ；

③ 用  $P(t)$  进行移动变异生成  $M(t)$ ；

④ 采样从  $P(t)$ 、 $S(t)$ 、 $C(t)$ 、 $M(t)$  中选出最好的不重复的下一代染色体:

$t = t + 1$

END

## 2、改进的 DNA 进化算法

改进的 DNA 进化算法中引入了交换操作（交换操作就是在 DNA 单链中随意产生一个位置，然后将位置前的 DNA 链与位置后的 DNA 链相交换，组成一条新的链）以更好地搜索解空间，并采用黄金分割率控制变异个体的数目。同时为了进一步提高搜索性能，采用一种新颖的启发式规则。

具体算法如下：对于每个工件都有 3 个时间指数： $t_{aj}$  为工件  $j$  在所有机器上的加工时间之和； $t_{1j}$  为工件  $j$  在第一台机器上的加工时间； $t_{mj}$  为工件  $j$  在最后一台机器上的加工时间； $t_j$  为工件  $j$  的加权加工时间。 $B$ 、 $C$  是  $[0, 1]$  之间的数。当随机生成一个  $A$ ，再在  $[0, 1-A]$  之间随机产生一个  $B$  便能确定  $t_j$  的大小，然后每个工件按照  $T_j$  的降序排列，这样就会产生一个可行解。生成不同的  $A$ ，就会得到不同的可行解。将启发式算法得到的可行解作为 DNA 进化算法的初始群体。具体算法如下：

① 计算每个工件  $t_{mj}$  的及  $t_{1j}$ ；

② for(  $I = 1, 2, \dots, n$ ) ( $n$  表示要产生的可行解的个数)；

$A = \text{random}(0, 1)$ ；

$B = \text{random}(0, 1-A)$ ；

$t_{ij} = At_{ij} + Bt_{1j} + (1 - A - B)t_{mj}$ ；

End

③ 根据每个工件计算出的  $t_j$  进行降序排列。得到对应的工件排序，即可行解。通过仿真可以验证，加入启发式算法能够快速接近最优解，提高算法的收敛速度，产生初始种群。

### 3、一种基于遗传算法的求解方法

一种基于遗传算法的求解方法，在由染色体转换成可行调度的过程中引入工件插入方法，同时设计了一种新的交叉算子（这里设计了一种新的交叉算子，从种群中按交叉概率随机选取两个个体作为父体，对于每个个体随机寻找两个不同的基因位置，选择这两个位置及其之间的基因作为交叉部分，两个交叉部分的长度可以不同。首先将两个交叉部分进行交换，然后按照父体中原来基因排列的顺序补齐交叉部分没有包含的基因，经过交叉之后产生的子代个体一部分基因保留了在一个父辈个体中的绝对位置，另一部分基因则保留了在另一个父辈中的相对位置。该操作具有较好的遗传特性，同时也能够产生足够的搜索空间。计算表明该算子优于 PMX 交叉算子。）通过大量的数值计算表明，该算法优化质量大大优于传统的遗传算法和 NEH 启发式算法。

### 4、一个无等待流水车间调度启发式算法

采用一个经典的全局任务插入算法构造初始解，应用局部搜索方法对其进行改进。通过 4000 个不同规模实例将提出算法与目前求解该问题最好的几个算法从性能和计算时间方面进行全面比较。实验结果表明：提出算法的性能是目前最好的，多项式复杂度的计算时间适合实际生产需求。此启发式算法包括两个阶段：初始序列的产生阶段和改进阶段。

(1) 在初始序列的产生阶段，采用任务插入的方法，它类似于 NEH[3] 算法。

(2) 在初始序列的改进阶段，定义  $v=(x, y)$  为序列  $S$  中的一对位置，其中： $x, y \in \{1, 2, \dots, n\}$ ,  $x \neq y$ 。v 的移动将  $S$  中第  $x$  个任务插入到第  $y$  个位置，位置对集合： $Z = \{(x, y) | x, y \in \{1, 2, \dots, n\}, y \in \{x, x-1\}\}$ ，其中包括  $(n-1)(n-1)$  个位置对。算法描述如下：

①令  $k=1$ ，计算所有任务  $j_i$  ( $j = 1, 2, \dots, n$ ) 的  $F2$  值。选择最小值对应的任务放入  $S$  中，将其余  $n-1$  个任务放入  $R$  中；

② $K = K + 1$ ;

③从  $R$  中任意取出一个任务  $j$ ，将其插入到  $S$  的  $k$  个不同位置，产生

k 个不同的序列，计算这 k 个序列的 F1 值，选择最小值对应的序列作为一个候选序列，将任务 j 从 R 中移除；

④如果 R 不为空集，返回第 3 步，否则转到第 5 步；

⑤在产生的  $(n - K + 1)$  个候选序列中，计算各自的 F 值，选择最小值对应的序列替换 S. 将序列 S 以外的所有任务存放到集合 R 中；

⑥如果  $K = n$ ，结束，S 即为最终初始序列；否则回到第 2 步继续；

⑦生成序列 S 的位置对集合并插入操作，产生  $(n - 1)(n - 1)$  个新的任务序列，计算所有新产生序列的 F1 值，将最小值对应序列记为 S'；

⑧如果  $F1(S') = F1(S)$ ，则  $S = S'$ ，返回第 7 步重新开始，否则转入第 9 步；

⑨序列 S 即为最终任务序列。

## 5、混合禁忌搜索算法(HTS)

(1) 混合禁忌搜索 HTS 算法的主要思路为：

通过一个有效的启发式算法为 TS 算法提供一个较好初始解，并可加快 TS 算法的收敛速度；采用禁忌搜索算法改进初始解以搜索到更好的近优解。初始解生成算法：

① 任意产生一个初始序列  $Q_0$ ；

② 利用双插入启发式算法[5](DIH)对序列 Q 进行改进获取一个序列  $S_n$ 。 DIH 基于全局插入操作和局部插入操作的思想来产生局部种子序列并对当前调度进行改进。该算法具有较高效率的搜索能力，得到一个较好的近优解；

③ 将序列  $S_n$  进行一次全局成对交换，得到初始序列 P。

(2) HTS 算法描述：

基于已得到的序列 P 作为初始解 T 和以上禁忌搜索算法，关键参数的设置，下面给出 HTS 算法：

① 调用初始解生产算法产生初始解 P 并赋予  $T_0$ ；

② 将初始解 T 作为当前解利用成对交换(Swap) 产生的邻域结构得

到多个邻域解；

③ 将所有邻域解对应的目标函数值从小到大排序，然后选取前  $e$  个邻域解作为候选解；

④ 从第 1 个候选解开始，如果满足藐视准则，则将此邻域解作为当前的序列  $T_1$ ；否则在候选解中选非禁忌的最佳状态序列作为当前序列  $T_1$ ；

⑤ 保存每个当前序列  $T_1$ ，及其目标函数值，并找出其中最优的目标函数值及对应的序列  $W_1$ ；

⑥ 若满足终止条件，则比较最后得到的当前序列  $T_1$  与序列  $W_1$  所对应的目标函数值大小，选取目标函数值小的序列作为算法最终所得到的近优解，算法停止；若不满足终止条件则  $T_0 = T_1$ ，则转向 2。

## 6、混合规划

针对不确定条件下流水车间调度问题 (Flow shop scheduling)，研究了含有随机参数和灰色参数的混合机会约束规划模型的建立及求解方法。提出了灰色模拟的概念和方法，为含有灰色参数的机会约束规划提供了求解途径。通过理论推导及仿真实例，结合遗传算法，验证了基于随机模拟和灰色模拟的混合机会约束规划的调度模型及求解方法的有效性。

## 三、 应用方案\应用系统\实验设置和结果展示（已有方法，改进的方法等，以及实验分析部分）（30 分）

使用模拟退火算法可以很好的求得流水车间调度问题的较优解。首先我们来介绍模拟退火算法。

模拟退火算法是一种模拟自然界物理过程的算法。通过模仿固体从高温徐徐冷却，内部分子从无序逐渐变成有序，内能减为最小的过程，使目标函数的值也达到最小。也就是说，我们可以设置一个变量来表示温度，而温度随循环不断减小，便是降温的过程。大量研究和实践保证了算法的较优性质，在收敛时有可观的效果。对于流水车间调度问题，这里针对目标函数特殊设计了计算的方法，利用了工件加工的有序性，前驱性，将约束条件转化为了 DAG 图上的边，通过遍历图的方式即可得出每个状态对应目标函数的值。



具体而言，我们可以按照以下几步实现模拟退火算法。

步骤一：设置一个初始温度  $T$ ，一个结束温度零温  $T_0$ ，还有指数衰减率  $rate$

步骤二：如果初始温度  $T \leq$  零温  $T_0$ ，则算法结束，将目前最优解作为答案

步骤三：随机产生一个当前状态邻域内的新状态，并用 DAG 评价函数估值

步骤四：如果新状态更优则接受；否则取两个状态估值的差  $\delta = new\_value - old\_value$ ，以指数分布的概率接受新状态

步骤五：温度下降，转到步骤二

接下来我们介绍一下如何具体处理流水车间调度问题的状态。

首先我们可以对于每一台设备  $i$  都确定一个该设备的加工顺序  $schedule_i$ ，其中  $schedule_{i,j}$  表示设备  $i$  加工的第  $j$  个工件。

其次很显然的是，如果我们确定了每一台设备中所有工件的加工顺序，那么这种情况下加工完所有工件所需的最短时间就是确定的，最短时间的求解方式如下：

对于  $n$  个工件和  $m$  台设备（即步骤），我们可以将其抽象为  $nm$  个点，每个点  $V_{i,j}$  代表由第  $i$  台设备加工第  $j$  个工件的任务。显然这些任务是有依赖关系的，而这些依赖关系可以转换为一张有向无环图。建图方式如下：

- a、对于同一台设备  $i$ ，其加工序列中相邻的两个工件  $schedule_{i,j}$  和  $schedule_{i,j+1}$  对应的任务  $V_{i,schedule_{i,j}}$  向  $V_{i,schedule_{i,j+1}}$  连一条有向边，表示同一台设备的加工列表中的先后顺序。
- b、对于同一工件  $j$ ，显然要由  $V_{i,j}$  向  $V_{i+1,j}$  连一条有向边，表示同一个工件的加工步骤先后顺序。

在该 DAG 上做 topo 排序并通过动态规划的方法计算关键路径长度，则到达  $V_{m,n}$  的关键路径长度与  $m$  设备加工  $n$  设备的时间之和即为当前状态下的最短加工时间。

为了满足模拟退火初始阶段的随机性，我们需要将某一个设备加工序列中的随机两个位置的工件序号交换，重复交换  $n*m$  次后开始进行退火过程。

在每次降温之后，我们需要得到一个由当前状态产生的新状态，产生方式如

下:

随机一个设备,再随即两个位置,交换该设备加工序列的这两个位置的工件标号即可。

重复上述过程直至温度将为 0。

下面给出算法伪代码:

```
function Simulated-Annealing(problem, schedule) returns a solution
state
```

```
inputs: problem, a problem
```

```
      T, highest temperature
```

```
      T0, lowest temperature
```

```
      rate, temperature's down rate
```

```
local variables: current, current state
```

```
               next, next state
```

```
current ← Random-initialization
```

```
while T>T0 begin
```

```
    next ← a randomly selected successor of current
```

```
    DAG-evaluate next
```

```
    delta ← new-value - old-value
```

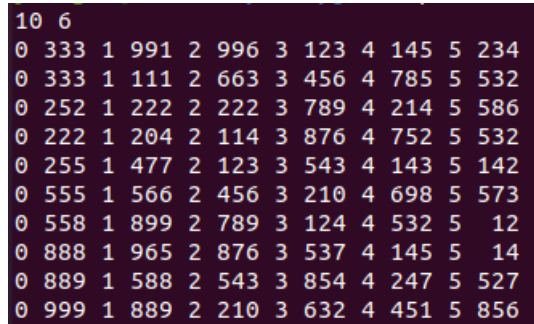
```
    if delta>0 then current ← next
```

```
    else current ← next only with probability  $\exp(\text{delta}/T)$ 
```

```
    T ←  $T*(1-\text{rate})$ 
```

```
end
```

实际执行效果如下:



```
10 6
0 333 1 991 2 996 3 123 4 145 5 234
0 333 1 111 2 663 3 456 4 785 5 532
0 252 1 222 2 222 3 789 4 214 5 586
0 222 1 204 2 114 3 876 4 752 5 532
0 255 1 477 2 123 3 543 4 143 5 142
0 555 1 566 2 456 3 210 4 698 5 573
0 558 1 899 2 789 3 124 4 532 5 12
0 888 1 965 2 876 3 537 4 145 5 14
0 889 1 588 2 543 3 854 4 247 5 527
0 999 1 889 2 210 3 632 4 451 5 856
```

图 1

分别有 10 个工件和 6 台设备，对应加工时长如下图 10\*12 的矩阵所示。我们将其输入到程序中后，每降温 1000 次输出温度和当前状态的最小花费：

```
-----Init End-----
Temperature: 904.846014 minCost: 22027
Temperature: 818.738122 minCost: 21404
Temperature: 740.824518 minCost: 18039
Temperature: 670.325409 minCost: 19825
Temperature: 606.535209 minCost: 15800
Temperature: 548.815478 minCost: 15220
Temperature: 496.588532 minCost: 16617
Temperature: 449.331660 minCost: 16234
Temperature: 406.571896 minCost: 16352
Temperature: 367.881281 minCost: 18225
Temperature: 332.872582 minCost: 12682
Temperature: 301.195417 minCost: 13328
Temperature: 272.532747 minCost: 11639
Temperature: 246.597704 minCost: 14444
Temperature: 223.130718 minCost: 11115
Temperature: 201.896922 minCost: 13889
Temperature: 182.683798 minCost: 11739
Temperature: 165.299054 minCost: 10281
Temperature: 149.568694 minCost: 11824
Temperature: 135.335283 minCost: 10812
Temperature: 122.456367 minCost: 10821
Temperature: 110.803048 minCost: 9889
Temperature: 100.258693 minCost: 9335
Temperature: 90.717772 minCost: 9187
Temperature: 82.084793 minCost: 8239
Temperature: 74.273355 minCost: 8835
Temperature: 67.205278 minCost: 8119
Temperature: 60.809819 minCost: 8854
Temperature: 55.022972 minCost: 8311
Temperature: 49.786819 minCost: 8095
Temperature: 45.048955 minCost: 8446
Temperature: 40.761959 minCost: 8107
Temperature: 36.882928 minCost: 8095
Temperature: 33.373036 minCost: 8198
Temperature: 30.197157 minCost: 8212
Temperature: 27.323504 minCost: 8116
Temperature: 24.723316 minCost: 8095
Temperature: 22.370571 minCost: 8095
Temperature: 20.241719 minCost: 8100
Temperature: 18.315456 minCost: 8107
Temperature: 16.572501 minCost: 8100
Temperature: 14.995412 minCost: 8095
Temperature: 13.568403 minCost: 8095
Temperature: 12.277193 minCost: 8107
Temperature: 11.108858 minCost: 8103
Temperature: 10.051705 minCost: 8097
Temperature: 9.095154 minCost: 8095
Temperature: 8.229632 minCost: 8095
Temperature: 7.446475 minCost: 8095
Temperature: 6.737846 minCost: 8095
Temperature: 6.096652 minCost: 8095
Temperature: 5.516476 minCost: 8095
Temperature: 4.991512 minCost: 8095
```

图 2

```

Temperature: 0.184100 minCost: 8095
Temperature: 0.166580 minCost: 8095
Temperature: 0.150728 minCost: 8095
Temperature: 0.136384 minCost: 8095
Temperature: 0.123405 minCost: 8095
Temperature: 0.111662 minCost: 8095
Temperature: 0.101036 minCost: 8095
Temperature: 0.091421 minCost: 8095
Temperature: 0.082721 minCost: 8095
Temperature: 0.074849 minCost: 8095
Temperature: 0.067726 minCost: 8095
Temperature: 0.061281 minCost: 8095
Temperature: 0.055449 minCost: 8095
Temperature: 0.050173 minCost: 8095
Temperature: 0.045398 minCost: 8095
Temperature: 0.041078 minCost: 8095
Temperature: 0.037169 minCost: 8095
Temperature: 0.033632 minCost: 8095
Temperature: 0.030431 minCost: 8095
Temperature: 0.027535 minCost: 8095
Temperature: 0.024915 minCost: 8095
Temperature: 0.022544 minCost: 8095
Temperature: 0.020399 minCost: 8095
Temperature: 0.018457 minCost: 8095
Temperature: 0.016701 minCost: 8095
Temperature: 0.015112 minCost: 8095
Temperature: 0.013674 minCost: 8095
Temperature: 0.012372 minCost: 8095
Temperature: 0.011195 minCost: 8095
Temperature: 0.010130 minCost: 8095
-----Sch Info-----
Machine 1 : 4 5 3 2 1 10 6 7 9 8
Machine 2 : 4 5 2 3 1 10 6 7 9 8
Machine 3 : 4 3 2 1 10 5 6 7 9 8
Machine 4 : 4 2 3 1 10 6 5 7 9 8
Machine 5 : 2 4 3 10 5 6 1 9 7 8
Machine 6 : 2 4 3 10 1 6 9 5 7 8
cost : 7876
6883574.000000

```

图 3

最终得到的最小代价为 7876。执行的总时长为 6.88s。最终的方案也如最后输出所示。

分析算法时间复杂度。可以看到算法的主体是一个循环，设循环执行了  $k$  次，则有  $T_0 = T(1 - rate)k$ ，可得  $k = \log(1 - rate) \frac{T_0}{T}$ ，同时对于 DAG-evaluate 函数，建图复杂度  $O(NM)$ ，进行拓扑序动态规划复杂度  $O(NM)$ ，

故总体复杂度为  $O(NM \log(1 - rate) \frac{T_0}{T})$ ，远低于寻找准确最优解所需的指数级复杂度。

分析空间复杂度。我们仅在 DAG-evaluate 函数中和状态的存储中使用到了一部分内存，并且复杂度均为  $O(NM)$ ，故在空间复杂度上也是可以接受的。

#### 四、 思考、自己的见解、改进的基本想法、结论（20 分）

根据实验结果我们可以得到以下模拟算法的优缺点：

1. 可以突破爬山算法的局限性，获得全局最优解（以一定的概率接受较差解，从而跳出局部最优解）

2. 初始解与最终解都是随机选取的，它们毫无关联，因此具有很好的鲁棒性，即抵御外界不稳定因素的能力。

3. 其最优解常常受迭代次数  $k$  的影响，若  $k$  值越大，则搜索时间越长。获得的最优解更可靠； $k$  值越小，则搜索时间越短，有可能就跳过了最优解。

4. 模拟退火算法受温度冷却速率的影响，若冷却速率较慢，则搜索时间较长，可以获得更优的解，但是会花费大量时间；如冷却速度过快，可能较快的搜索到更优的解，但也有可能直接跳过最优解。

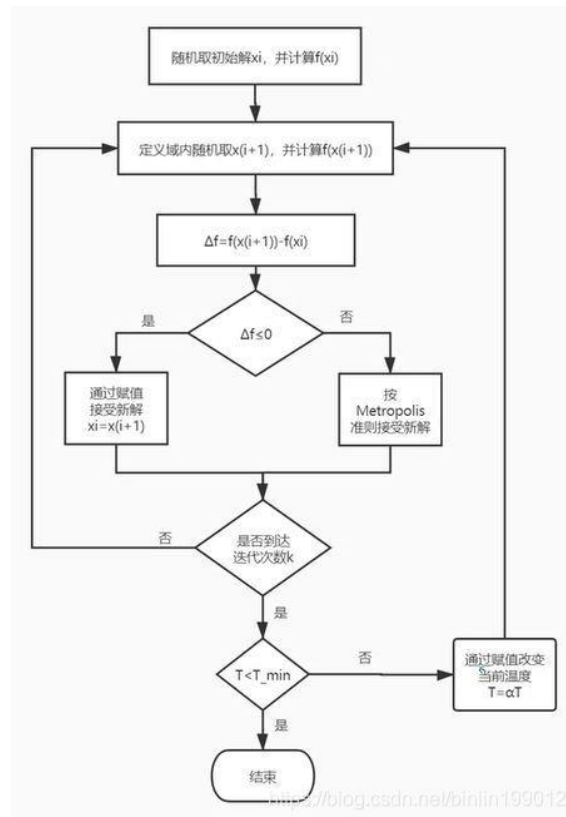


图 4

模拟退火算法的应用很广泛，可以高效地求解 NP 完全问题，如货郎担问题 (Travelling Salesman Problem, 简记为 TSP)、最大截问题 (Max Cut Problem)、0-1 背包问题 (Zero One Knapsack Problem)、图着色问题 (Graph Colouring Problem) 等等，但其参数难以控制，不能保证一次就收敛到最优值，一般需要多次尝试才能获得（大部分情况下还是会陷入局部最优值）。观察模拟退火算法的过程，发现其主要存在如下三个参数问题：

### (1) 温度 T 的初始值设置问题

温度 T 的初始值设置是影响模拟退火算法全局搜索性能的重要因素之一、初始温度高，则搜索到全局最优解的可能性大，但因此要花费大量的计算时间；反之，则可节约计算时间，但全局搜索性能可能受到影响。

### (2) 退火速度问题，即每个 T 值的迭代次数

模拟退火算法的全局搜索性能也与退火速度密切相关。一般来说，同一温度下的“充分”搜索是相当必要的，但这也需要计算时间。循环次数增加必定带来计算开销的增大。

### (3) 温度管理问题

温度管理问题也是模拟退火算法难以处理的问题之一。实际应用中，由于必须考虑计算复杂度的切实可行性等问题，常采用如下所示的降温方式：

$$T = \alpha \times T, \alpha \in (0,1)$$

注：为了保证较大的搜索空间， $\alpha$ 一般取接近于 1 的值，如 0.95、0.9。

本代码中为了尽量减少局部最优值的影响，会重复进行多次降温过程，每次降温的初始阶段为上一次退火过程的结果。

```
//尝试多个epoch可能找到更优解
for (int epoch = 1; epoch <= 5; epoch++) sa.SA();
```

图 5

流水线问题是模拟退火算法的一个较为具象的应用。实际上，模拟退火算法已经在包括图像处理、VLSI 设计、神经网络等多个领域实现应用。

## 五、 参考文献（10 分）

[1]冯爱芬,闻博卉,黄宇.基于模拟退火算法仓内拣货的优化问题[J].洛阳理工学院学报(自然科学版),2021,31(04):73-77.

[2]曲媛,杨晓伟.关于流水车间调度问题的综述[J].中小企业科技,2007(8):24-25.

[3]包波. A 公司自动化总装车间 AGV 路径优化问题研究[D].北京交通大学,2020.DOI:10.26944/d.cnki.gbfju.2020.002468.

[4]Ribas Imma,Companys Ramon,Tort-Martorell Xavier. An iterated greedy algorithm for the parallel blocking flow shop scheduling problem and sequence-depe

ndent setup times[J]. Expert Systems With Applications,2021,184:

[5]曹瑞瑞,孔建寿.改进 GA 算法求解自动开票流水线生产调度问题[J].工业控制计算机,2021,34(06):4-7+9.

[6]王艺霖,郑建国.离散蝙蝠算法在三阶段装配流水线调度问题的应用[J].控制与决策,2021,36(09):2267-2278.DOI:10.13195/j.kzyjc.2020.0054.