# 组合优化大作业

**冯浩哲**

**31040104019**

**统计1401**

**研究性问题C**

---



# 问题叙述

我选择的问题是**研究性问题C**中**最坏情况界下界**的改进问题。研究性问题C是一个求总装箱时间和最小的装箱问题，这个装箱问题也可以看成是1台机器加工不同工作量而相同加工时间的工件的并行等分问题。

具体来说，此时有一批等待加工的工件$J=\{J_1,J_2,\dots,J_n\}$，它们的加工时间都为单位时间，每个工件有1个参数，即需要的机器负荷$P=\{P_1,P_2,\dots,P_n\}$。仅有1台负载为$T$的机器$M$，它可以同时加工若干个工作量之和小于机器负载的工件，每个工件的加工时间都为单位时间，而我们的目的就是求总完工时间最小。为讨论方便起见，不失一般性我们令$P_i <= 1, \forall i \in \{1,2,\dots,n\}$以及$T=1$.

按机器每一次同时加工的工件将$J$分为若干个小集合，记作

$$J=\{\{J_{11},J_{12},\dots,J_{1k_1}\},\dots,\{J_{m1},J_{m2},\dots,J_{mk_m}\}\}$$

$$s.t. \forall i \in \{1,\dots,m\}, \sum_{j=1}^{k_i} P_{ij} \le T$$

那么按照上面的叙述，我们的总完工时间可以用一个简单的公式如下表述：

$$\min \sum_{i=1}^{n} C_i = \sum_{i=1}^{m} i * k_i$$

# 已有结论回顾与扩展

**文献[1]**中给出了上述问题的两个结论:

1. 用**FFI**求解$\min \sum_{i=1}^{n} C_i$问题的最坏情况界$r \leq \frac{2+\sqrt{2}}{2}$
2. 可以构造实例满足$r \geq \frac{109}{82}$, $\frac{109}{82} \leq 1.32927$

而我在本次作业中相对第二个结论进行改进，即构造一组实例$J, P$，满足最坏情况界$\geq \frac{109}{86}$.

# 成果与解决方案

## 基本思路回顾

我采用了启发式算法找到了增加FFI算法最坏情况下界的实例，具体而言，我采用遗传算法寻找的这个实例满足让FFI算法与重排序FFD(**EXFFD**算法,将在**文献实例探索**部分叙述)所得$\sum_{i=1}^{n} C_i$之比$\frac{C^{FFI}(J)}{C^{EXFFD}(J)}$大于$\frac{109}{82}$.

在整个过程中我需要解决以下几个问题:

1. 如何选取近似算法使得在**FFI**算法表现差的时候该近似算法表现很好

   我最后选择使用了**EXFFD**算法，并在**文献实例探索**部分叙述了原因。

2. 采用什么样的实例与近似算法构造原则可以让**FFI**算法所得的$C^{FFI}(J)$尽量大，同时令$C^{FFD}(J)$尽量小。

   我在**文献实例探索**部分会叙述并证明2个构造原则，这是找到实例的关键。

3. 如何确定优化空间与优化算法

   在优化算法方面，我首先测试了模拟退火算法，之后测试了遗传算法。在确定优化空间方面，我一开始确定的优化空间是固定工件总数目而寻找一组工件需要的机器负荷参数$P = \{P_1, P_2, \ldots, P_n\}$，但是这种策略表现并不好。我与本课程的**刘一璟**同学讨论了这个问题（刘一璟同学也采用了遗传算法构造实例的思路来解决问题，并获得了比我更好的结果），他建议我选取一组$P$，而优化每一个负荷参数下的工件数目，并为这个优化空间提供了一些参考的初值。我采用了这个建议并找到了实例，而这个建议被证明在整个体系中是**至关重要**的，特此致敬。

## 成果展示

我找到了两组实例，使得它在$FFI$算法下的最坏情况界大于$\frac{109}{82}$.这两组实例(分别记为$I, II$)如下所示:

| $(I)$工件数目 | $(I)$工件尺寸 | $(II)$工件数目 | $(II)$工件尺寸 |
|---|---|---|---|
| 6 | 0.1 | 6 | 0.1 |
| 1 | 0.19 | 1 | 0.19 |
| 1 | 0.29 | 1 | 0.3 |
| 13 | 0.39 | 12 | 0.39 |
| 7 | 0.4 | 10 | 0.4 |
| 3 | 0.41 | 8 | 0.41 |
| 3 | 0.49 | 1 | 0.49 |
| 2 | 0.5 | 18 | 0.51 |
| 18 | 0.51 | 7 | 0.59 |
| 6 | 0.59 | 14 | 0.6 |
| 7 | 0.6 | 15 | 1 |
| 10 | 1 | | |
| **最坏情况界** | $\geq 1.3298$ | **最坏情况界** | $\geq 1.32974$ |

## 文献实例探索

针对最坏情况界下界，文献中给出了一组实例如下

| 工件数目 | 工件尺寸 |
|---|---|
| 1 | 0.1 |
| 2 | 0.2 |
| 2 | 0.3-eps |
| 1 | 0.4-eps |
| 10 | 0.5-eps |
| 13 | 0.5+eps |
| 8 | 1 |

通过对这组实例进行分析，我发现这组实例之所以能导致比较大的最坏情况界下界，一个很重要的原因是在$0.5-eps$之前的物体用FFI时已经自己结对按顺序在单台机器上加工了，这就使得$0.5+eps$的物体只能一个一个加工，这样就导致了较坏情况的出现。同时，在这个实例下，箱子被浪费的空间非常大。也就是说，这个情况糟糕的原因在于先加工小物体导致数目比较大的大物体只能单独依次加工。那么一个很自然的想法是，在FFI情况不好的

时候，FFD可以加工尽量少的次数就完成工作，但是这是否意味着FFD的总完工时间也相对更少呢?

在这个情况中，我验证了在FFD算法下的总完工时间，发现计算的结果为486,大于FFI所得的436，但是FFD确实用了尽可能少的盒子，我对这个情况进行了分析，并得出了本文第一个构造实例原则。

上实例按FFD算法加工顺序如下：

| 加工组合 | 组合加工次数 |
| --- | --- |
| {1} | 8 |
| {0.5-eps,0.5+eps} | 10 |
| {0.5+eps,0.4-eps,0.1} | 1 |
| {0.5+eps,0.3-eps,0.2} | 2 |

从该结果中我们可以发现，原因是工件尺寸为1的工件被预先加工了，这样就耽误了后面的工件时间。因此与$1||\sum C_j$的最优解原则相似，这里最优解的一个原则叙述如下：

## 最优解构造原则1

> 假设工件集$J$在加工过程中可以分成若干个集合：
>
> $$J = \{\{J_{11}, J_{12}, \dots, J_{1k_1}\}, \dots, \{J_{m1}, J_{m2}, \dots, J_{mk_m}\}\}$$
> $$= \{S_1, S_2, \dots, S_m\}$$
> $$s.t. \forall i \in \{1, \dots, m\}, \sum_{j=1}^{k_i} P_{ij} \leq T$$
>
> 其中存在$p, q \in \{1, 2, \dots, m\}, s.t. |S_p| < |S_q|$,那么我们将$S_p, S_q$的顺序调换不会增大总完工时间。

这是一个显然的结论，因为调换集合$S_p, S_q$后，这若干个集合的依次完工时间$\{t_1, t_2, \dots, t_m\}$并不会改变，因此总完工时间就减少了

$$(|S_q| - |S_p|) * (t_q - t_p)$$

按照上面的条件，这是一个正数。我们按该原则调换完工时间所得加工顺序如下：

| 加工组合 | 组合加工次数 |
| --- | --- |
| {0.5+eps,0.4-eps,0.1} | 1 |
| {0.5+eps,0.3-eps,0.2} | 2 |
| {0.5-eps,0.5+eps} | 10 |
| {1} | 8 |

以该顺序所得的总完工时间为328。记录调换顺序后的**FFD**算法为**EXFFD**,因此通过调换顺序，$C^{FFI}(J)/C^{EXFFD}(J)$就达到了文献[1]中所给的最坏实例下界，而这也表明，**调换过加工顺序后的FFD算法在FFI算法表现不好的时候表现优异，其结果可以作为近似最优解，因此** $\frac{C^{FFI}(J)}{C^{EXFFD}(J)}$**是一个寻找最坏情况下界的一个好的估计。**

同时，我们注意到最优解构造的另一个问题是，在遵循原则1的情况下，对机器的利用率越高(即机器工作时的最小负载$L_{min}$尽可能大)，是否意味着$\sum_{i=1}^{n} C_i$更少呢？这里我给出一个构造，证明这个结论在某些实例上是不存在的，该构造如下表所示：

| 工件数目 | 工件尺寸 |
| --- | --- |
| 18 | eps |
| 19 | 0.1-eps/2 |
| 2 | 0.2 |
| 2 | 0.3-eps |
| 19 | 0.4-eps/2 |
| 18 | 0.5-2*eps |
| 39 | 0.5+eps |
| 0 | 1 |

该实例在FFI下的$\sum_{i=1}^{n} C_i$结果为2130,如果我们达到最大利用率，我们可以构造39个集合，每个集合都是由3个工件组成的，总负担为1，在这个构造下结果为

$$\sum_{i=1}^{39} 3i = 2340 < 2130$$

这个悖论表示，选取使得机器利用率最大的算法得到的结果不一定是好的。这就有了第二个最优解构造原则

### 最优解构造原则2

注意到该悖论形成的原因是因为有大量P值非常小的工件，而同时大工件的数目相对变少(如该悖论实例中尺寸为1的工件为0)，这就导致FFI算法将用2次就把这些P值非常小的大量工件完成了，但是最大利用率算法将把这些小工件塞在较大工件的空隙中完成，这增大了小工件的完工期。因此，在已知**EXFFD**算法能够在**FFI**算法的机器利用率很低($L_{min}$很大)的时候取得较高的利用率这一条件时，我们需要遵循以下原则，使得$L_{min}$上的优势可以转换为$\sum_{i=1}^{n} C_i$上的优势。

> 在选取实例：工件$J=\{J_1,J_2,\dots,J_n\}$以及机器负荷$P=\{P_1,P_2,\dots,P_n\}$的过程中，我们应该尽量减少机器负荷值较少的工件数目而增加机器负荷较大的工件数目

## 优化算法与优化对象选取

### 优化对象选取

首先，我将工件的总数目看作一个固定值（算法中取了40），以每个工件的机器负荷P为优化对象。在优化过程中，我的阈值始终停留在**1.30**附近而无法获得进一步增长。我认为这是因为优化对象区间过大的原因。同时这个优化目标也没有体现我上文的实例构造原则2。

正如前文对**刘一璟**同学的致敬，我最终固定了机器负荷参数，选取了每个负荷参数对应的工件数目为优化目标。为了体现原则2，同时利用文献[1]中构造的实例所用的思想，我选取了如下的机器负荷参数,该组参数与文献[1]构造实例的**Size of Jobs**一栏非常相似，我同时使用0.1代替了$\epsilon$,参数如下：

```
Size_of_Jobs=
[0.1, 0.11, 0.19, 0.2, 0.29, 0.3, 0.31, 0.39, 0.4, 0.41, 0.49, 0.5, 0.51, 0.59, 0.6, 0.61, 0.69,
1]
```

## 优化算法选取

在选取优化算法的过程中，我尝试了模拟退火算法与遗传算法[2]两个算法，我将叙述两个算法的具体流程与结果，并重点探讨我最后选择使用的遗传算法。

1. 模拟退火算法

   文献[2]已经叙述了模拟退火算法的基本思路，我仅简要叙述应用细节以伪代码的形式如下：

```
# Set the temperature parameter T and decay parameter r
T=1e6;r=0.9999
# Random choose a series of numbers according to Size_of_Jobs
Number_of_Jobs=[number for number in Uniform(0,50) and number is an integer]
# Calculate the ration of FFI(Number_of_Jobs)/EXFFD(Number_of_Jobs)
ratio=calculate_ratio(Number_of_Jobs)
# check if ratio is bigger than our target
while ratio < 109/82:
    # Random change every element of Number_of_Jobs using the index
  for index in len(Number_of_Jobs):
    # Create a new Number_of_Jobs
    New_Number_of_Jobs=copy(Number_of_Jobs)
    # Add random change
    new_Number_of_Jobs[index] += random_integer
    # Calculate Score
    new_ratio=calculaye_ratio(new_Number_of_Jobs)
    # if new ratio is bigger than old, accept new, else accept new with a probability
    if new_ratio > ratio:
      Number_of_Jobs=copy(new_Number_of_Jobs)
      ratio=new_ratio
    else:
      # choose an accept probability
      accept_probability =exp(-1*abs(new_ratio-ratio)/(T))
      set Number_of_Jobs=copy(new_Number_of_Jobs),ratio=new_ratio with accept_probability
  # change T and simulate annealing progress
  T=T*r
```

   我在使用该算法的时候发现算法收敛性并不好，并且无法逃脱局部极小值(受困于1.27)，这使我想到了遗传算法。

2. 遗传算法

文献[2]已经叙述了模拟退火算法的基本思路，我简要叙述应用细节以伪代码的形式如下，其中遗传算法用到的变异过程与基因交叉过程在算法后面详细叙述。

```
# Set the group size, the top select rate and the mutation probability
group_size=50;top_select_rate=0.1,mutation_prob=0.3
# choose group_size's number of random integers from 0 to 50
Group=[]
# repeat random choose 50 times
for i in range(group_size):
    # Random choose a series of numbers according to Size_of_Jobs
    Number_of_Jobs=[number for number in Uniform(0,50) and number is an integer]
    # Add to group
    Group.append(Number_of_Jobs)
# calculate ratio for each element in group
ratios=calculate_ratio(group)
# find max ratio
maxratio=max(ratios)
# find if we reach the target
while maxratio < 109/82:
  # do mutation with mutation_prob and do intersection with 1-mutation_prob
  if do_mutation:
    # random select one from group
    item=random_select_one(Group)
    # do random increase or decrease for every element
    for element in item:
      element=element+ random_integer from -3 to 3
    ratio_mutation=calculate_ratio(item)
    if ratio_mutation > min(ratios):
      #move the min(ratios) and the related item
      group[argmin(ratios)]=item
      #add the new item to group
  if do_intersection:
    # select the top items and do intersection in them
    top_Group=select_top_rate(Group,ratios)
    # random select two in top_Group
    item1,item2=random_select_two(top_Group)
    # choose one cut integer randomly
    item_intersection=item1[0:cut_integer]+item2[cut_integer:]
    ratio_intersection=calculate_ratio(item_intersection)
    if ratio_intersection > min(ratios):
      #move the min(ratios) and the related item
      group[argmin(ratios)]=item_intersection
      #add the new item to group
  # re-calculate ratio
  ratios=calculate_ratio(group)
  maxratio=max(ratios)
```

注意到在交叉过程中我的算法采用的是concat操作，而此操作得到了**可行实例($I$)**.

其实concat操作减少了运算量，但是它在运行的过程中很容易收敛到局部极值，因此我也尝试了另一种操作，即给定两个待杂交可行解$solution1, solution2$,采用如下加权平均操作:

```
solution_intersection=w*solution1+(1-w)*solution2
```

这种操作运算量较大，但是它容易脱离局部极值，此操作得到了**可行实例($II$)**.

# 解决方案

以遗传算法为核心，我找出了如成果展示的2组可行实例。这里我会将**FFI**算法与**FFD**算法的分配具体罗列出来，并给出它们的时间比。

## 可行实例$I$

| FFI Solution | FFI Solution | EXFFD Solution | EXFFD Solution |
|---|---|---|---|
| 1. [0.1 (*6), 0.19] | 9. [0.5] | 1. [0.51,0.39,0.1] *6 | 9. [1] *10 |
| 2. [0.29,0.39] | 10. [0.51] *18 | 2. [0.51,0.29,0.19] | 10 [0.51] *4 |
| 3. [0.39] *6 | 11. [0.59] *6 | 3. [0.6,0.4] *7 | |
| 4. [0.4,0.4] *3 | 12. [0.6] *7 | 4. [0.59,0.41] *8 | |
| 5. [0.4,0.41] | 13 [1] *10 | 5. [0.59,0.39] *3 | |
| 6. [0.41,0.41] | | 6. [0.51,0.49] *3 | |
| 7. [0.49,0.49] | | 7. [0.51,0.39] *4 | |
| 8. [0.49,0.5] | | 8. [0.5,0.5] | |
| $\sum C_i$ | 1778 | $\sum C_i$ | 1337 |

## 可行实例$II$

| FFI Solution | FFI Solution | EXFFD Solution | EXFFD Solution |
|---|---|---|---|
| 1. [0.1(*6) ,0.19] | 9. [0.51] *18 | 1. [0.51,0.39,0.1] *6 | 9. [1] *15 |
| 2. [0.3,0.39] | 10. [0.59] *7 | 2. [0.51,0.3,0.19] | 10. [0.51] *7 |
| 3. [0.39,0.39] * 5 | 11. [0.6] *14 | 3. [0.6,0.4]  *10 | |
| 4. [0.39,0.4] | 12. [1] *15 | 4. [0.6,0.39] *4 | |
| 5. [0.4,0.4] *4 | | 5. [0.59,0.41] *7 | |
| 6. [0.4,0.41] | | 6. [0.51,0.49] *1 | |
| 7. [0.41,0.41] *3 | | 7. [0.51,0.41] | |
| 8. [0.41,0.49] | | 8. [0.51,0.39] *2 | |
| $\sum C_i$ | 2714 | $\sum C_i$ | 2041 |

# 参考文献

[1] Li, R, Tan, Z, Zhu, Q. Minimizing Total Completion Time of Batch Scheduling with Non-identical Job Sizes. Proceeding of the 9th International Conference on Combinatorial Optimization and Applications, Lecture Notes in Computer Science, 10627, 165-179

[2]Segaran T. Programming collective intelligence: building smart web 2.0 applications[M]. " O'Reilly Media, Inc.", 2007,95-97