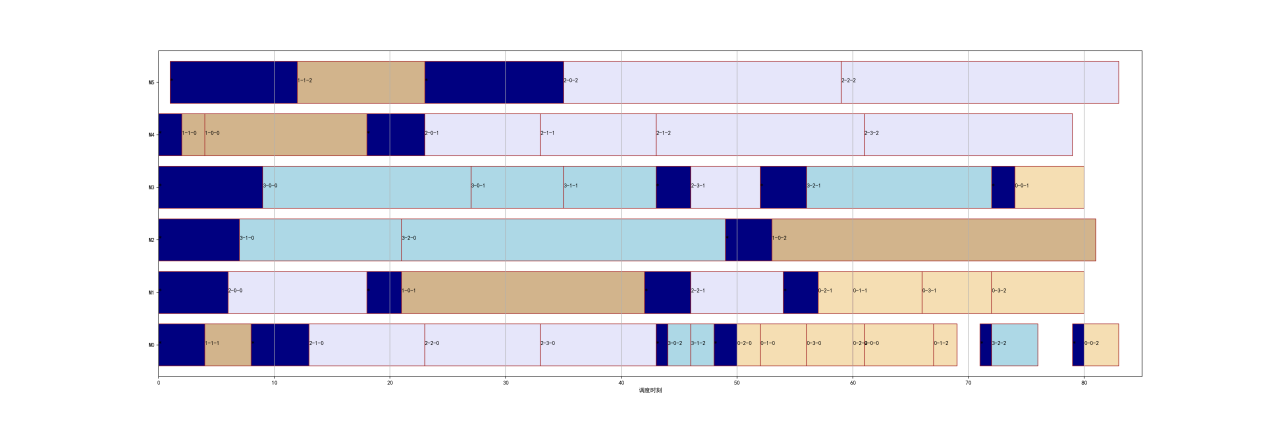
# 编码方式

## 实验结果：

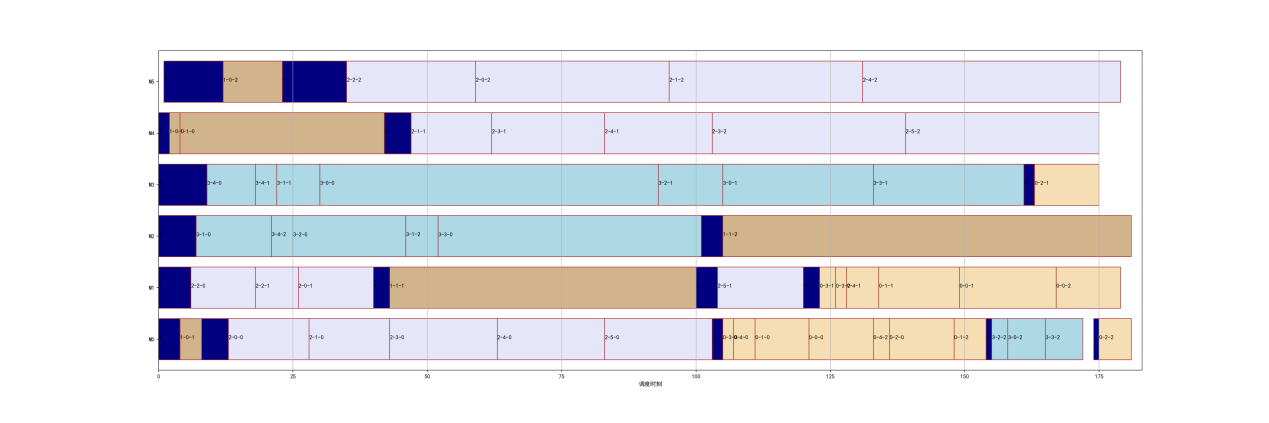
我的编码方式能找到的最好的解

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 问题 | 别人的最小完工时间 | 我的最小完工时间 | gant数据 | 编码 |
| P1 | 85 | 83 | bestgant-P1.csv | [4, 2, 4, 3], [[3, 2, 1, 2], [7, 1], [2, 2, 2, 2], [2, 2, 4]], [[1, 2, 3, 0], [1, 2, 3, 0], [3, 1, 0, 2], [3, 1, 0, 2], [2, 0, 3, 1], [1, 2, 3, 0]] |
| P2 | 183 | 181 | bestgant-P2.csv | [5, 2, 6, 5], [[6, 5, 6, 1, 2], [1, 19], [3, 3, 2, 4, 4, 4], [7, 2, 3, 7, 1]], [[1, 2, 0, 3], [1, 2, 0, 3], [3, 1, 0, 2], [3, 0, 2, 1], [3, 2, 0, 1], [2, 1, 3, 0]] |
| P3 | 213 | 192 | bestgant-P3.csv | [5, 3, 4, 3, 6, 6], [[1, 2, 1, 3, 3], [4, 5, 1], [2, 2, 2, 4], [4, 4, 2], [1, 1, 2, 2, 2, 2], [2, 2, 2, 1, 1, 2]], [[2, 3, 5, 0, 4, 1], [2, 1, 5, 0, 4, 3], [1, 2, 3, 4, 5, 0], [0, 1, 3, 5, 2, 4], [5, 4, 0, 2, 3, 1], [3, 5, 2, 4, 0, 1]] |
| P4 | 415 | 370 | bestgant-P4.csv | [7, 7, 8, 6, 8, 8], [[5, 2, 4, 1, 2, 1, 5], [3, 2, 3, 3, 4, 1, 4], [4, 4, 1, 2, 3, 4, 1, 1], [4, 4, 4, 2, 2, 4], [1, 2, 3, 5, 2, 5, 1, 1], [2, 2, 3, 3, 3, 1, 3, 3]], [[4, 0, 3, 2, 1, 5], [0, 1, 2, 3, 4, 5], [1, 3, 4, 5, 0, 2], [2, 0, 1, 5, 3, 4], [4, 5, 0, 3, 2, 1], [1, 3, 4, 5, 0, 2]] |
| P5 |  | 125 | bestgant-P5.csv | [4, 4, 4], [[2, 2, 2, 2], [1, 3, 2, 2], [2, 2, 2, 2]], [[1, 0, 2], [1, 2, 0], [0, 2, 1], [2, 0, 1], [2, 0, 1], [2, 0, 1], [1, 2, 0], [2, 1, 0]] |
| P6 |  | 216 | bestgant-P6.csv | [5, 5, 6], [[3, 3, 3, 3, 3], [3, 3, 3, 3, 3], [3, 1, 3, 3, 3, 2]], [[2, 0, 1], [1, 0, 2], [1, 0, 2], [2, 0, 1], [0, 1, 2], [0, 1, 2], [0, 2, 1], [1, 0, 2]]) |
| P7 |  | 188 | bestgant-P7.csv | [4, 3, 4, 4, 4], [[2, 2, 2, 2], [2, 3, 3], [2, 2, 2, 2], [2, 3, 1, 2], [2, 2, 2, 2]], [[3, 0, 2, 1, 4], [3, 1, 4, 0, 2], [1, 2, 0, 4, 3], [4, 3, 0, 1, 2], [1, 0, 3, 4, 2], [1, 2, 0, 4, 3], [2, 0, 4, 1, 3], [0, 3, 2, 4, 1]]) |
| P8 |  | 337 | bestgant-P8.csv | [5, 6, 5, 6, 6], [[3, 3, 3, 3, 3], [2, 2, 3, 3, 3, 2], [3, 3, 3, 3, 3], [3, 3, 3, 2, 2, 2], [3, 3, 3, 3, 1, 2]], [[4, 2, 0, 3, 1], [3, 0, 1, 2, 4], [2, 0, 3, 1, 4], [4, 0, 3, 2, 1], [1, 3, 2, 4, 0], [0, 2, 3, 4, 1], [0, 3, 2, 1, 4], [0, 2, 3, 1, 4]]) |

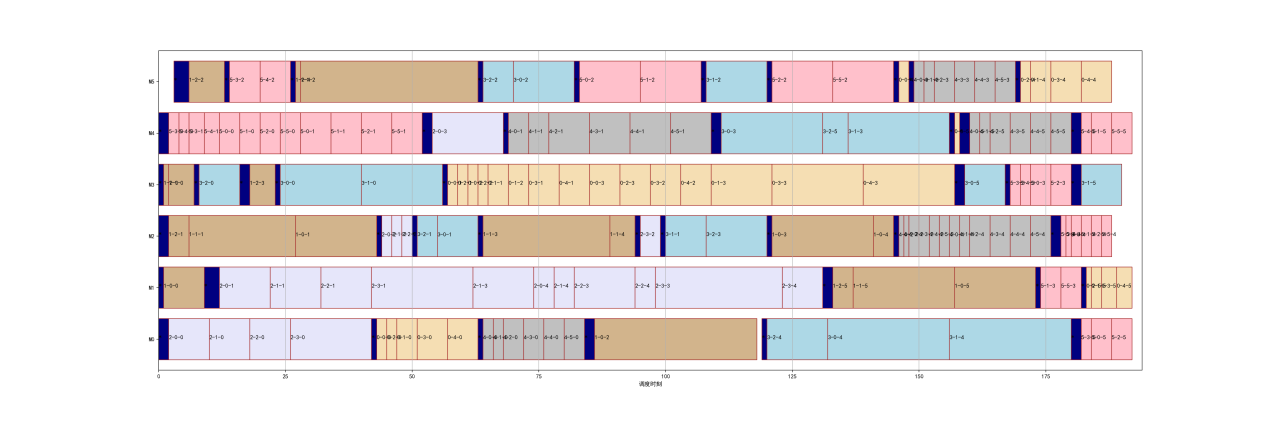
P1：



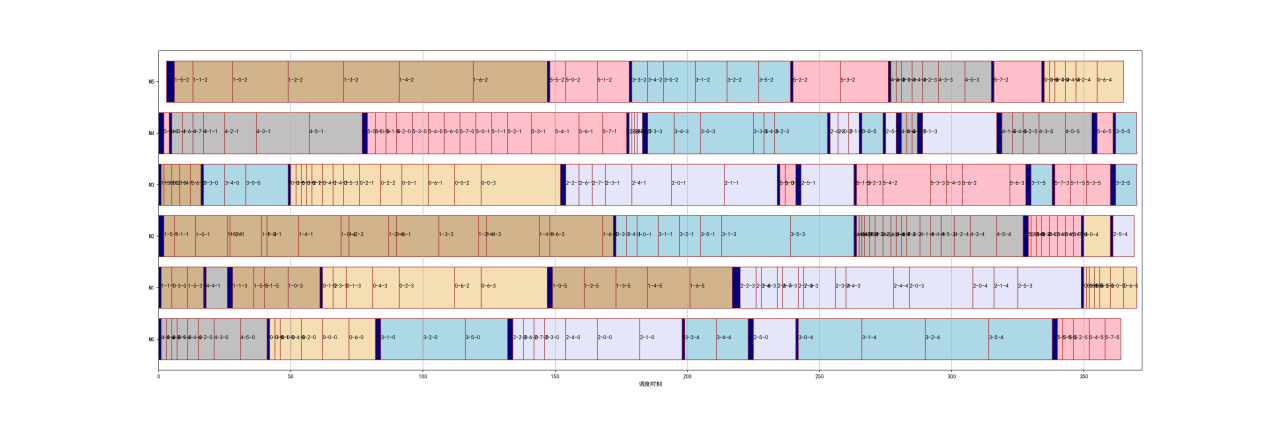
P2：



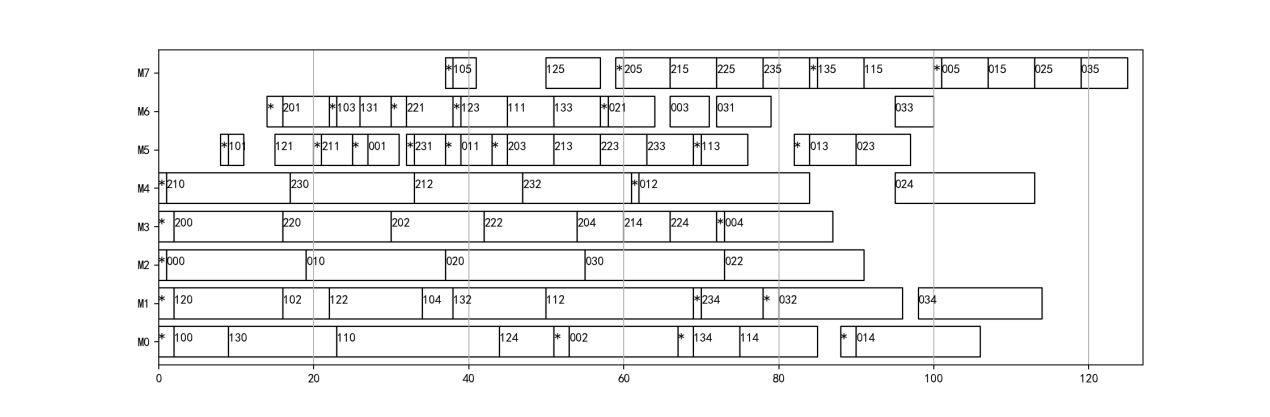
P3:



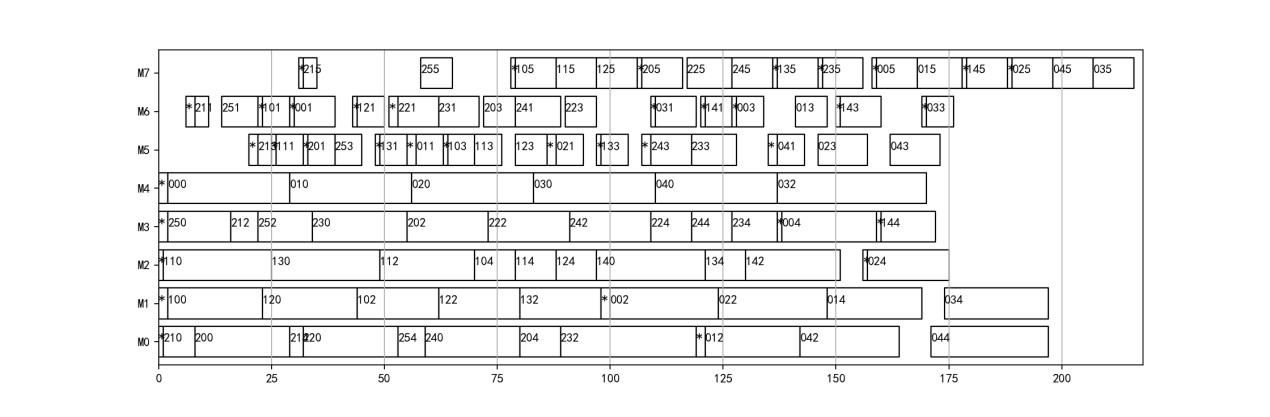
P4：



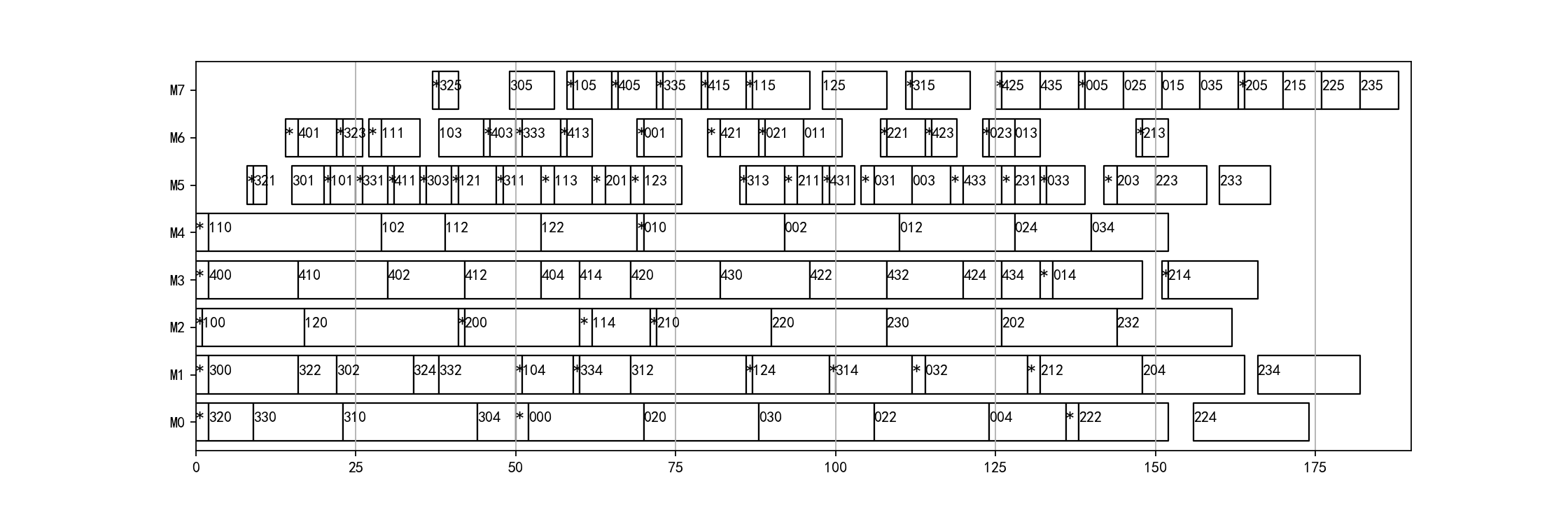
P5：



P6：



P7：



# 其他算法

## 数据结果

### IMGA

原始的岛群GA

参数：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| outerIterNum | innerIterNum | p1 | p2 | p3 | ps1 |
| 100 | 10 | 0.8 | 0.3 | 0.3 | 0.4 |
| ps2 | ps3 | ps4 | ps5 | mode | migratePercentage |
| 0.4 | 0.3 | 0.3 | 0.3 | ‘exchange’ | 10 |

原始数据：

|  |  |
| --- | --- |
| P1 | 85, 85, 85, 83, 85, 85, 85, 89, 85, 86, 85, 86, 85, 86, 85, 85, 85, 85, 85, 85 |
| P2 | 185, 183, 187, 188, 187, 185, 187, 185, 185, 186, 185, 185, 187, 188, 189, 189, 189, 189, 190, 185 |
| P3 | 195, 198, 197, 199, 197, 197, 194, 195, 195, 196, 194, 193, 194, 199, 195, 196, 197, 198, 194, 195 |
| P4 | 378, 377, 381, 372, 380, 377, 378, 373, 382, 375, 378, 378, 379, 376, 376, 376, 370, 377, 375, 379 |

### NIMGA

来自An effective new island model genetic algorithm for job shop scheduling problem

P2~p3都改为了0.5，要重新实验。改的目的是，细粒度的改变只能完全依靠mutation。而不能依赖交叉，所以mutation的概率要适当提高

参数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| iterNum | islandSize | p1 | p2 | p3 |
| 1000 | 18 | 0.8 | 0.3 | 0.3 |

原始数据：

|  |  |
| --- | --- |
| P1 | 85, 89, 85, 89, 83, 88, 86, 87, 85, 88, 92, 88, 85, 87, 83, 90, 89, 86, 89, 85 |
| P2 | 196, 192, 193, 195, 190, 191, 184, 185, 197, 185, 189, 199, 183, 193, 185, 193, 203, 189, 189, 188 |
| P3 | 202, 204, 209, 207, 203, 200, 207, 200, 206, 206, 201, 202, 199, 201, 208, 206, 199, 202, 202, 198 |
| P4 | 387, 383, 385, 391, 384, 391, 383, 384, 384, 380, 379, 392, 392, 386, 394, 381, 394, 396, 396, 386 |

### HGA

来自 A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop

### MABC

来自Scheduling job shop with lot streaming and transportation through a modified artificial bee colony

# 原始MBO

## 参数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| iteration | popsize | K | S | M |
| 1000 | 51 | 3 | 1 | 10 |

## 特性：

1. 只有细粒度邻域算子，没有粗粒度的

2. 没有启发式的邻域算子

## 感性分析：

1. 很容易陷入局部最优，例如p=2时，整个种群就陷入了194，p=3时，整个种群就陷入了209。

2. 进化非常缓慢，可能是因为只用了细粒度邻域搜索

## 数据结果：

原始数据

|  |  |
| --- | --- |
| P1 | 86, 87, 84, 83, 88, 86, 89, 85, 85, 86, 83, 85, 87, 83, 84, 85, 85, 83, 84, 83 |
| P2 | 188, 190, 186, 194, 185, 192, 185, 189, 189, 186, 187, 185, 187, 188, 194, 195, 200, 194, 184, 185 |
| P3 | 197, 200, 201, 199, 198, 197, 196, 198, 204, 202, 199, 207, 198, 201, 199, 199, 200, 207, 201, 202, |
| P4 | 389, 385, 392, 378, 385, 381, 391, 385, 382, 389, 392, 384, 378, 390, 383, 385, 385, 378, 390, 388 |

统计数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| problem | Running times | mean | std | best | worse | Times of BK | BK |
| 1 | 20 | 85.5 | 1.7168284 | 83 | 89 | 5 | 83 |
| 2 | 20 | 189.15 | 4.234087 | 184 | 200 | 0 | 181 |
| 3 | 20 | 200.25 | 2.9474565 | 196 | 207 | 0 | 192 |
| 4 | 20 | 385.5 | 4.4553338 | 378 | 392 | 0 | 370 |

# 原始MBO+新邻域算子（启发式&粗粒度）

## 参数：

（除了额外的邻域算子以外，其他与原始MBO相同）

|  |  |
| --- | --- |
| S1coarse range | S2coarse range |
| 0.13 | 0.13 |

## 各个邻域算子起作用的次数：

单次run结果：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | S1n1 | S1n2 | S1n3 | S2n1 | S2n2 | S2n3 | S2n4 | S1coarse | S2coarse |
| P1 | 28 | 24 | 30 | 18 | 15 | 30 | 24 | 41 | 24 |
| P2 | 65 | 69 | 49 | 18 | 21 | 17 | 25 | 67 | 21 |
| P3 | 58 | 51 | 66 | 72 | 47 | 30 | 40 | 74 | 31 |
| P4 | 102 | 77 | 76 | 58 | 48 | 58 | 45 | 50 | 42 |

20run结果：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | S1n1 | S1n2 | S1n3 | S2n1 | S2n2 | S2n3 | S2n4 | S1coarse | S2coarse |
| P1 | 609 | 605 | 621 | 339 | 355 | 382 | 399 | 800 | 383 |
| P2 | 1492 | 1145 | 1010 | 426 | 443 | 499 | 564 | 896 | 553 |
| P3 | 924 | 951 | 1053 | 920 | 800 | 836 | 752 | 1132 | 790 |
| P4 | 1777 | 1541 | 1359 | 1059 | 924 | 887 | 886 | 1328 | 811 |
| 总 | 4802 | 4242 | 4043 | 2744 | 2522 | 2604 | 2601 | 4156 | 2537 |

额外加的实验（10run）：

P2：{'s1n1': 712, 's1n2': 542, 's1n3': 444, 's2n1': 192, 's2n2': 227, 's2n3': 231, 's2n4': 223, 's1coarse': 420, 's2coarse': 208}

P3: {'s1n1': 487, 's1n2': 459, 's1n3': 501, 's2n1': 464, 's2n2': 411, 's2n3': 378, 's2n4': 378, 's1coarse': 533, 's2coarse': 368}

## 分析：

1. 在各个算子机会均等的情况下，S1的算子效果好于S2的算子，说明邻域搜索的重点在于S1，应该减少S2算子的数量。其中的原因是，S1的解空间比S2的大一些，可搜索范围大一些。

下面是平均单个算子在20run中成功的次数：

|  |  |  |
| --- | --- | --- |
|  | S1 | S2 |
| P1 | 2635/4=658.75 | 1858/5=371.6 |
| P2 | 4543/4=1135.75 | 2485/5=497 |
| P3 | 4060/4=1015 | 4096/5=819.6 |
| P4 | 6005/4=1501.25 | 4567/5=913.4 |
| 总 | 17243/16=1077.6875 | 13006/20=650.3 |

2. S1n1效果不错，在S1算子里排第1，说明加入启发式有效果。S1coarse效果也不错，排第3，比S1n3好，说明粗粒度有效。

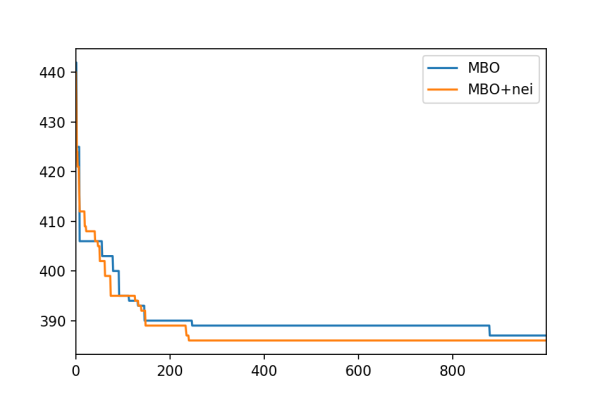
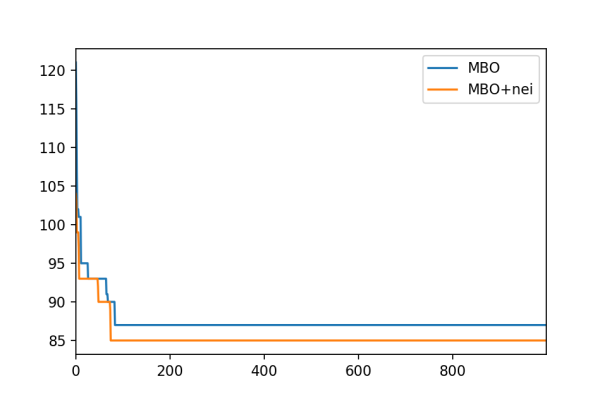
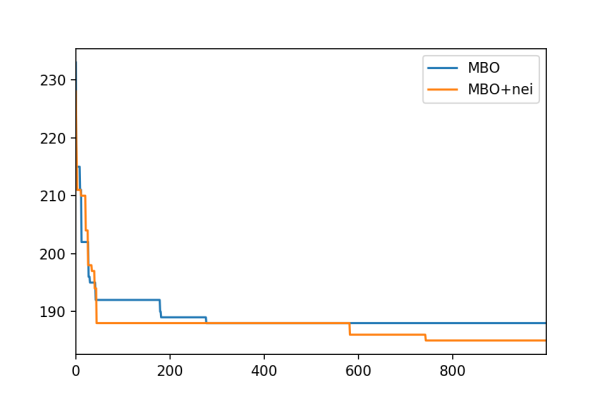
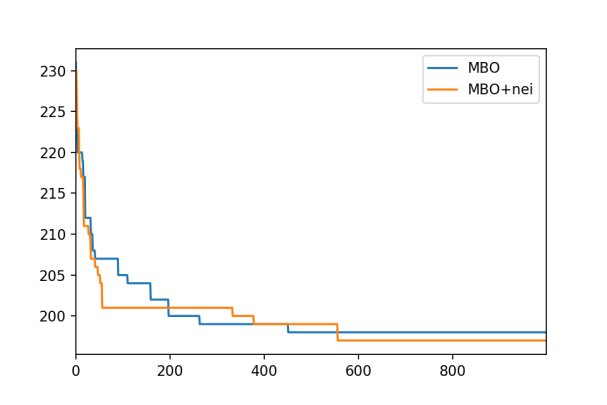
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | S1n1 | S1n2 | S1n3 | S1coarse |
| 次数 | 4802 | 4242 | 4043 | 4156 |
| 排名 | 1 | 2 | 4 | 3 |

3. S2n1效果不错，在S2算子里排第1，说明加入启发式有效果。S2coarse效果也不错，排第4，比S2n2，说明粗粒度有效。

粗粒度在一定程度上能代替S2n2中insert的效果，而且S2n2效果是最差的，因此考虑不使用S2n2，留下4个S2算子。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | S2n1 | S2n2 | S2n3 | S2n4 | S2coarse |
| 次数 | 2744 | 2522 | 2604 | 2601 | 2537 |
| 排名 | 1 | 5 | 2 | 3 | 4 |

## 收敛曲线：



## 数据结果：

还在不断补充数据，后期要更新统计数据

原始数据

|  |  |
| --- | --- |
| P1 | 85, 85, 85, 83, 89, 88, 86, 89, 85, 87, 85, 87, 85, 85, 83, 83, 85,85, 85, 87 |
| P2 | 189, 184, 189, 184, 192, 191, 185, 185, 185, 191, 185, 193, 195,185, 196, 199, 193, 187, 189, 187, 183 |
| P3 | 195,194, 203, 196, 202, 203, 200, 204, 201, 198, 201, 200, 201, 199, 196,202, 204, 202, 202, 200, 201, 198, 202, 203, 196, 200, 197, 200, 202, 200 |
| P4 | 386, 401, 389, 389, 381, 387, 381, 377, 382, 384, 386, 388, 387,387, 379, 386, 383, 374, 383, 376 |

统计数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| problem | Running times | mean | std | best | worse | Times of BK | BK |
| 1 | 20 | 85.6 | 1.7146428 | 83 | 89 | 3 | 83 |
| 2 | 20 | 189.2 | 4.3081318 | 183 | 199 | 0 | 181 |
| 3 | 20 | 200.15 | 2.8857408 | 194 | 204 | 0 | 192 |
| 4 | 20 | 384.5 | 5.7192656 | 37 | 401 | 0 | 370 |

# myMBO1

## 参数：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| iteration | popsize | K | S | M | A |
| 1000 | 51 | 3 | 1 | 8 | 2 |

A是队形调整阶段的迭代次数

## 特性：

1. 邻域共享改为交叉

2. 加入模糊顺序V字型

3. 加入队形调整阶段

## 数据结果：

原始数据：

|  |  |
| --- | --- |
| P1 | 83, 85, 85, 83, 85, 85, 85, 85, 85, 85, 85, 85, 86, 86, 85, 85, 88, 85, 87, 88 |
| P2 | 189, 187, 187, 194, 183, 195, 188, 183, 192, 189, 189, 187, 189, 191, 184, 190, 183, 189, 187, 189 |
| P3 | 198, 200, 196, 199, 197, 196, 197, 197, 198, 196, 199, 199, 200, 198, 196, 197, 199, 199, 198, 196 |
| P4 | 380, 378, 374, 383, 380, 385, 381, 378, 378, 375, 382, 382, 386, 384, 385, 377, 377, 382, 384, 376 |

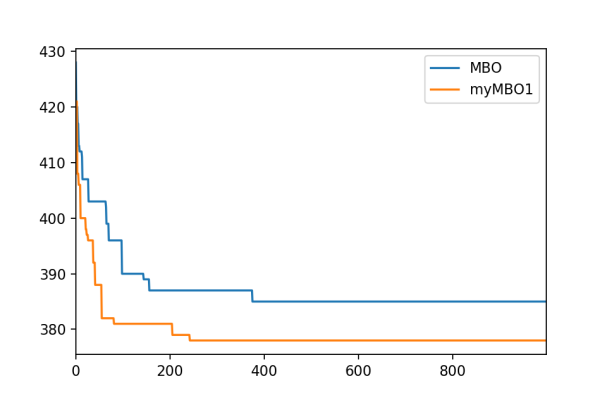
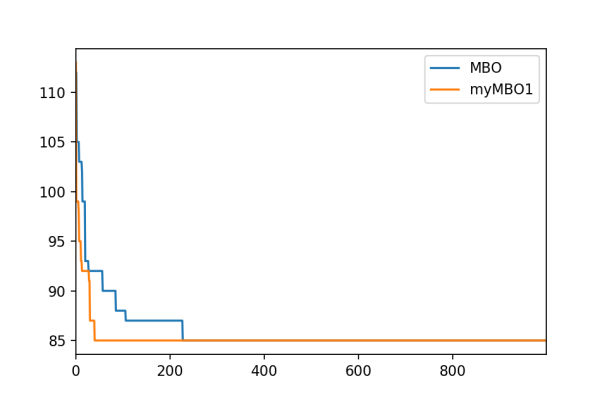
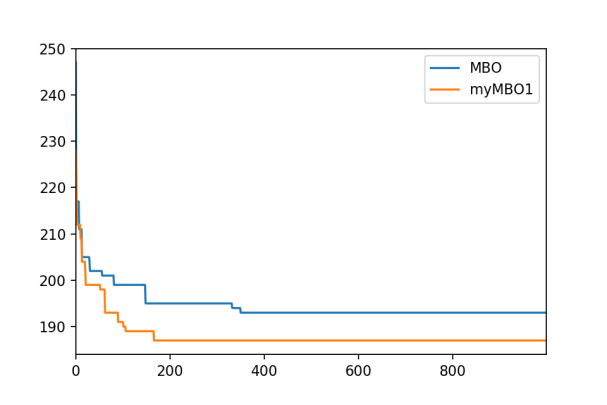
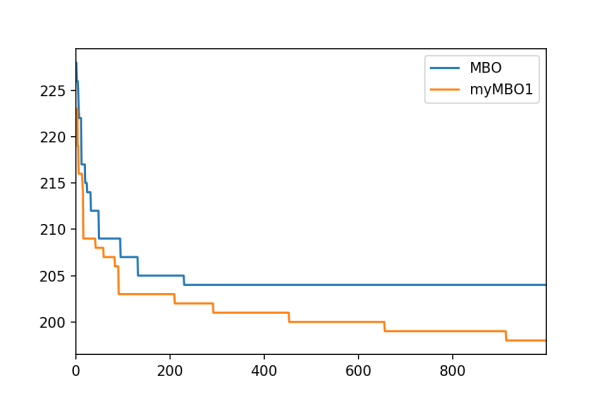
统计数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| problem | Running times | mean | std | best | worse | Times of BK | BK |
| 1 | 20 | 85.3 | 1.2288205 | 83 | 88 | 2 | 83 |
| 2 | 20 | 188.25 | 3.2691742 | 183 | 195 | 0 | 181 |
| 3 | 20 | 197.75 | 1.3369741 | 196 | 200 | 0 | 192 |
| 4 | 20 | 380.35 | 3.4967842 | 374 | 386 | 0 | 370 |

各个邻域成功次数（20run）：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | S1n1 | S1n2 | S1n3 | S2n1 | S2n2 | S2n3 | S2n4 | S1coarse | S2coarse |
| P1 | 472 | 520 | 558 | 283 | 299 | 341 | 339 | 764 | 293 |
| P2 | 1096 | 988 | 851 | 346 | 409 | 412 | 407 | 873 | 412 |
| P3 | 797 | 1016 | 1321 | 920 | 1046 | 968 | 929 | 1358 | 914 |
| P4 | 1469 | 1633 | 1524 | 1040 | 1077 | 1063 | 997 | 1399 | 841 |
| 总 |  |  |  |  |  |  |  |  |  |

## 收敛曲线：



# myMBO1+aging

## 参数：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| iteration | popsize | K | S | M | agingThreshold |
| 500 | 51 | 3 | 1 | 10 | [10, 30, 40] |

## 特性：

在myMBO1的基础上，加上aging策略：

age< agingThreshold[0]，用一次一步邻域算子

agingThreshold[0]<=age< agingThreshold[1]，用两次一步邻域算子

agingThreshold[1]<=age< agingThreshold[2]，用两次两步邻域算子

agingThreshold[2]<=age，重新初始化

## 数据结果：

从收敛图上看，500代还是少了点，后面要重新做实验

原始数据：

|  |  |
| --- | --- |
| P1 | 83, 83, 85, 85, 85, 85, 85, 83, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 83 |
| P2 | 183, 187, 183, 187, 185, 186, 183, 185, 185, 185, 183, 185, 189, 184, 185, 189, 184, 183, 183, 181 |
| P3 | 194, 195, 196, 193, 194, 193, 194, 194, 195, 194, 195, 195, 192, 195, 193, 194, 195, 194, 195, 194 |
| P4 | 371, 375, 370, 374, 373, 371, 375, 376, 375, 373, 374, 370, 374, 371, 373, 377, 376, 376, 374, 374 |

统计数据：

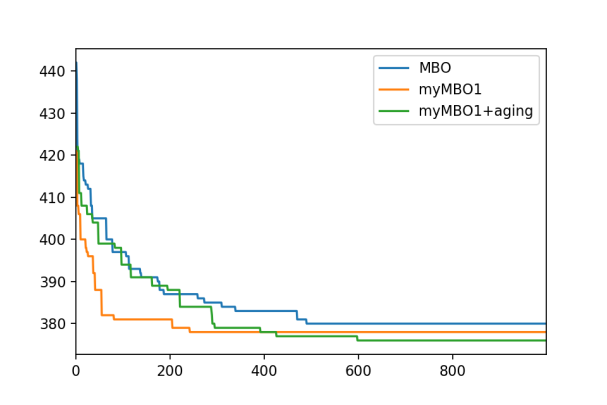
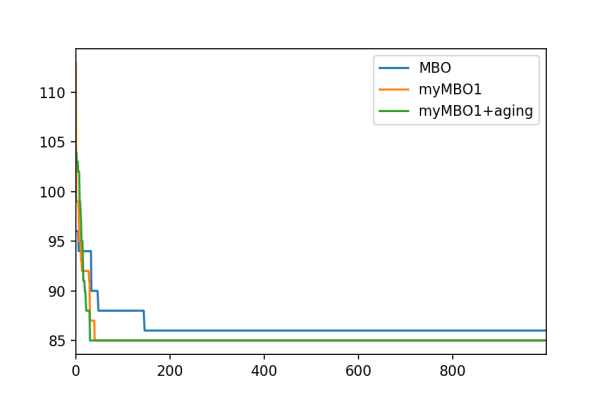
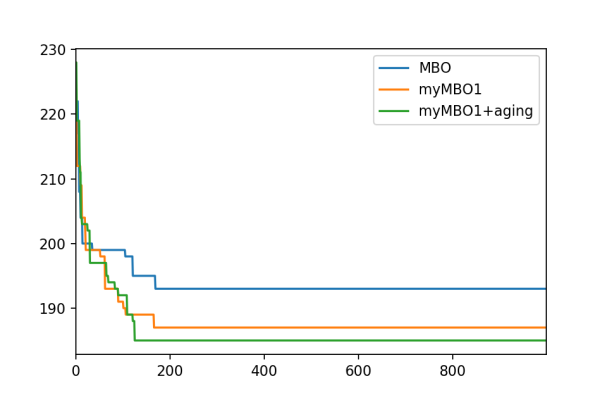
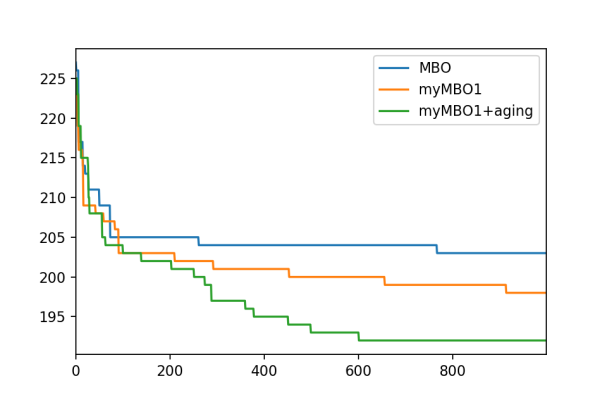
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| problem | Running times | mean | std | best | worse | Times of BK | BK |
| 1 | 20 | 84.6 | 0.7999999 | 83 | 85 | 4 | 83 |
| 2 | 20 | 184.75 | 2.0217566 | 181 | 189 | 1 | 181 |
| 3 | 20 | 194.2 | 0.9273618 | 192 | 196 | 192 | 192 |
| 4 | 20 | 373.6 | 2.0346989 | 370 | 377 | 2 | 370 |

## Aging成功次数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 一次一步 | 两次一步 | 两次两步 | 重新初始化 |
| P1 | 12179 | 643 | 51 | 8241 |
| P2 | 15663 | 1257 | 113 | 7060 |
| P3 | 18846 | 3948 | 469 | 5507 |
| P4 | 21526 | 4694 | 506 | 5141 |

感觉还是改为成功百分比比较好，即成功次数/试探次数，这样才公平

## 收敛曲线：



# MyPMBO1 分化+退化

## 数据结果：

原始数据：

|  |  |
| --- | --- |
| P1 | 85, 85, 83, 85, 85, 85, 83, 85, 85, 85 |
| P2 | 186, 186, 184, 189, 189, 187, 186, 187, 187, 188, 185, 183, 185, 188, 185, 188, 185, 185, 189, 184 |
| P3 | 194, 195, 196, 194, 195, 195, 194, 195, 197, 194 |
| P4 | 377, 378, 375, 376, 373, 373, 376, 374, 373, 378 |

Aging改为了4段

|  |  |
| --- | --- |
| P1 |  |
| P2 | 188, 187, 185, 189, 185, 183, 187, 183, 183, 185, 185 |
| P3 |  |
| P4 |  |

Aging改为了4段+迁移百分比改为30（并没有好很多）

|  |  |
| --- | --- |
| P1 |  |
| P2 | 183, 184, 187, 189, 183, 185, 187, 184, 187, 184 |
| P3 |  |
| P4 |  |

Aging改为了4段+迁移百分比改为30+使用s1&s2邻域（效果有点不稳定，但有时出奇地好啊）

|  |  |
| --- | --- |
| P1 | 85, 83, 83, 85, 85, 85, 85, 85, 83, 83, 85, 85, 86, 85, 83, 85, 85, 85, 85, 83 |
| P2 | 185, 181, 187, 181, 185, 186, 185, 183, 185, 187, 188, 189, 185, 183, 183, 187, 183, 185, 189, 186 |
| P3 | 196, 194, 194, 195, 197, 193, 194, 192, 196, 195, 195, 195, 194, 195, 195, 191, 193, 193, 196, 193 |
| P4 | 375, 376, 376, 375, 372, 375, 377, 378, 375, 375, 376, 376, 378, 373, 377, 377, 378, 374, 376, 373 |

Aging改为了4段+迁移百分比改为30+使用s1&s2邻域+改popsize为29

|  |  |
| --- | --- |
| P1 | 85, 83, 85, 83, 85, 85, 85, 85, 85, 83, 85, 85, 85, 83, 85, 85, 85, 83, 83, 83 |
| P2 | 185, 183, 185, 189, 183, 183, 183, 186, 188, 185 |
| P3 | 193, 193, 193, 194, 194, 194, 193, 195, 193, 194, 194, 194, 195, 195, 193, 194, 194, 194, 193, 194 |
| P4 |  |

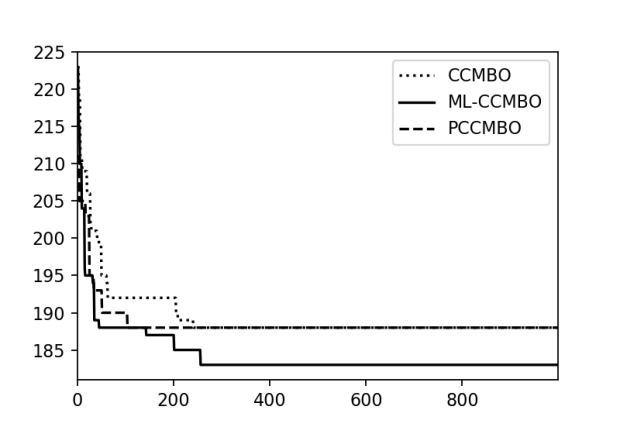
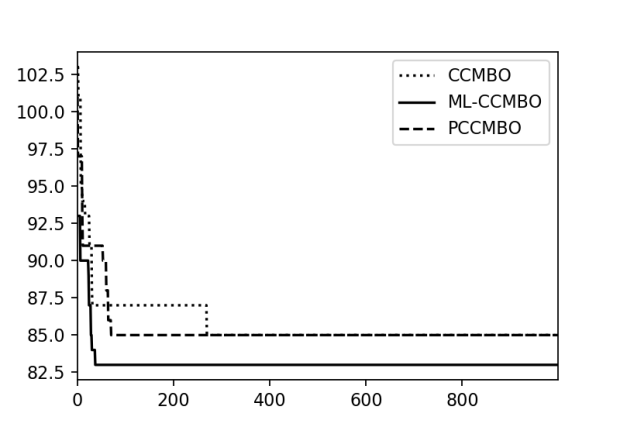
# 各种算法对比

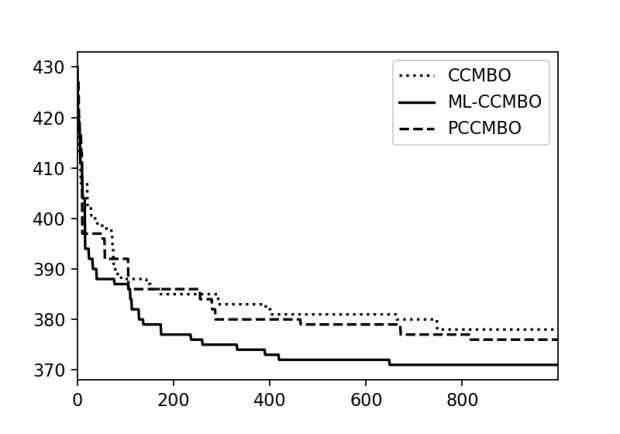
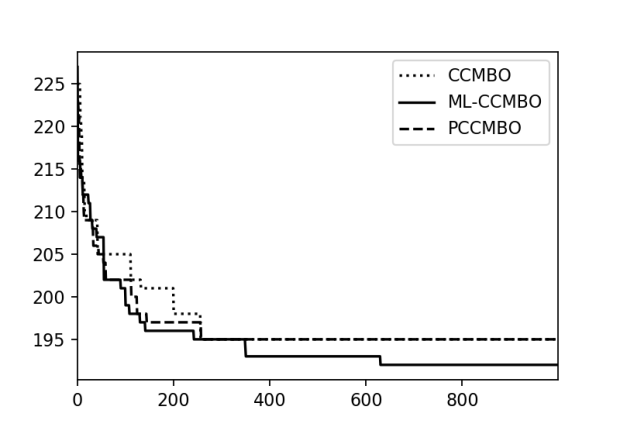
然后对CCMBO、CCMBO的简单的多种群化算法PCCMBO、本章提出的改进多种群算法ML-CCMBO进行对比。为了公平起见，三个算法所使用的种群个体总数都为51，而且都使用了相同的迭代次数。对算例P1~P8求解30次的结果如表所示。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **problem No.** | | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** |
| **CCMBO** | **Avg** | 85.13 | 188.10 | 197.87 | 380.73 | 125.80 | 217.90 | 192.07 | 340.10 |
| **Std** | 1.54 | 3.03 | 1.33 | 3.51 | 0.65 | 2.15 | 2.79 | 1.74 |
| **Best** | 83 | 183 | 196 | 374 | 125 | 215 | 188 | 337 |
| **Worst** | 88 | 195 | 200 | 386 | 127 | 224 | 196 | 343 |
| **BKN** | 7 | 4 | 0 | 0 | 10 | 1 | 5 | 4 |
| **PCCMBO** | **Avg** | 84.53 | 186.27 | 194.93 | 375.40 | 125.53 | 217.70 | 191.53 | 339.57 |
| **Std** | **0.85** | 2.29 | 1.18 | 2.29 | 0.50 | 1.72 | 2.72 | 1.50 |
| **Best** | 83 | 183 | **192** | **371** | 125 | 215 | 188 | 337 |
| **Worst** | **85** | 190 | 197 | 379 | **126** | 224 | 196 | 343 |
| **BKN** | 7 | 7 | 1 | 3 | 14 | 2 | 6 | 5 |
| **ML-CCMBO** | **Avg** | **84.00** | **184.37** | **193.57** | **373.10** | **125.33** | **216.23** | **188.67** | **338.20** |
| **Std** | 1.00 | **1.40** | **1.02** | **1.64** | **0.47** | **1.33** | **0.98** | **1.17** |
| **Best** | 83 | 183 | **192** | **371** | 125 | 215 | 188 | 337 |
| **Worst** | **85** | **187** | **195** | **376** | **126** | **219** | **191** | **340** |
| **BKN** | **15** | **13** | **7** | **7** | **20** | **11** | **19** | **14** |

PCCMBO是CCMBO的简单多种群化，从Avg来看，PCCMBO

|  |  |  |  |
| --- | --- | --- | --- |
| **problem No.** | **CCMBO** | **PCCMBO** | **ML-CCMBO** |
| **P1** | 1.17% | 0.84% | 0.55% |
| **P2** | 2.79% | 1.79% | 0.75% |
| **P3** | 3.06% | 1.53% | 0.82% |
| **P4** | 2.62% | 1.19% | 0.57% |
| **P5** | 0.64% | 0.43% | 0.27% |
| **P6** | 1.35% | 1.26% | 0.57% |
| **P7** | 2.16% | 1.88% | 0.35% |
| **P8** | 0.92% | 0.76% | 0.36% |
| **Avg** | 1.84% | 1.21% | 0.53% |





[93, 90, 91, 95, 100, 99, 85]

[2.6457513110645907, 2.981423969999719, 2.8674417556808756, 1.9720265943665385, 1.2909944487358056, 1.4142135623730951, 7.544313531837517]

EM [99, 89, 119, 111, 93, 105, 92, 96]

EM [1.8856180831641267, 6.904105059069326, 1.3743685418725535, 1.5723301886761003, 4.921607686744467, 1.699673171197595, 5.937171043518958, 3.131382371342656]

EM [91, 93, 89, 87, 97, 96, 103, 95]

EM [4.775516260631468, 3.2145502536643185, 8.132581932503902, 10.884494578170465, 1.5986105077709065, 2.748737083745107, 1.4624940645653537, 2.852873794770615]

EM [93, 90, 91, 95, 100, 99, 85]

EM [2.6457513110645907, 2.981423969999719, 2.8674417556808756, 1.9720265943665385, 1.2909944487358056, 1.4142135623730951, 7.544313531837517]

EM [92, 108, 95, 91]

EM [2.9860788111948193, 1.0671873729054748, 2.2110831935702664, 3.131382371342656]

EM [87, 104, 102, 91, 121, 92]

EM [11.026483271348727, 2.565800719723442, 2.9860788111948193, 8.117197107923948, 2.266911751455907, 3.144660377352201]

EM [85, 92, 93, 89]

EM [7.951240294584376, 5.185449728701348, 3.0550504633038935, 6.264982043070834]

EM [96, 97, 101, 108, 98]

EM [8.556998435328957, 3.7712361663282534, 3.144660377352201, 2.748737083745107, 3.7601713908928263]

先把所有能找到的最优解找出来

# 多目标算法结果

## P1

(103, 0.9574271077563381) ,(95, 1.3743685418725535), (88, 1.7950549357115013)

[(114, 1.4624940645653537), (113, 1.5986105077709065), (101, 2.753785273643051), (103, 2.2852182001336816), (104, 1.9790570145063195), (102, 2.581988897471611), (92, 3.131382371342656), (111, 1.795054935711501), (94, 2.8722813232690143), (90, 4.921607686744467)]

[(91, 2.266911751455907), (88, 2.4094720491334933), (102, 1.5986105077709065)]

EM [(94, 3.0776975521032313), (93, 3.815174380753199), (95, 2.5440562537456244), (88, 4.955356249106169), (96, 2.2110831935702664)]

EM [(94, 3.0776975521032313), (93, 3.815174380753199), (95, 2.5440562537456244), (88, 4.955356249106169), (96, 2.2110831935702664)]

EM [(91, 2.266911751455907), (88, 2.4094720491334933), (87, 4.810289896553937), (102, 1.4907119849998596)]

[(83, 1.3743685418725535)]

[(90, 2.494438257849294), (96, 1.1547005383792515), (98, 0.9574271077563381), (85, 5.163977794943222), (95, 1.5723301886761005), (89, 3.435921354681384), (92, 2.357022603955158), (86, 4.546060565661952), (87, 4.524623986832743)]

## P2

EM [(204, 6.182412330330469), (220, 1.9720265943665387), (222, 1.632993161855452), (201, 7.987837977547839), (215, 2.70801280154532), (205, 4.9916597106239795), (216, 2.362907813126304)]

EM [(195, 2.4324199198877374), (190, 5.112620550059322)]

EM [(208, 2.7688746209726918), (211, 1.5723301886761007), (200, 5.617433182117573), (202, 3.7155828016013257)]

EM [(209, 1.0671873729054748), (195, 2.6718699236468995), (193, 2.981423969999719), (200, 2.494438257849294), (204, 1.3844373104863459), (190, 4.272001872658765), (191, 4.268749491621898), (192, 4.179978734661484)]

EM [(221, 1.5986105077709065), (205, 1.6749792701868151), (190, 9.838981428763628), (194, 2.4094720491334933), (204, 2.034425935955617), (198, 2.217355782608345), (192, 5.676462121975467), (193, 3.1091263510296048)]

[(203, 2.3392781412697), (189, 2.362907813126304)]

[(195, 4.058598553961973), (200, 1.863389981249825), (204, 1.5275252316519468), (197, 2.9249881291307074), (188, 5.408326913195984), (190, 4.9805175991613115), (198, 2.581988897471611)]

## P3

(210, 6.128258770283411)

[(203, 2.1921577396609844), (208, 1.674979270186815)]

EM [(219, 0.7453559924999299), (207, 3.131382371342656), (254, 0.6871842709362768), (210, 1.8929694486000912), (216, 1.5723301886761007), (206, 3.337497399083464), (208, 3.095695936834452), (205, 5.619905100029122)]

## P4

(387, 2.852873794770615)

(385, 5.785518319236594)

(386, 2.6246692913372702)

(394, 2.6874192494328497)

(387, 5.909032633745278)

(395, 4.027681991198191)

(389, 7.11024300256718)

EM [(383, 1.4907119849998596)]

EM [(389, 5.82141639885766), (387, 6.289320754704403), (390, 5.033222956847166), (394, 2.3570226039551585), (393, 3.605551275463989), (397, 0.9428090415820632)]

# P1

### MMOMBO

[(88, 5.4594464513864), (89, 3.2489314482696545), (95, 1.5723301886761005)]

[(97, 1.0), (96, 1.5986105077709065), (85, 4.4969125210773475), (87, 4.374801582802228), (88, 3.7155828016013257), (90, 1.8929694486000912)]

[(102, 0.7453559924999299), (85, 5.163977794943222), (94, 2.4267032964268394), (108, 0.6871842709362768), (96, 0.9428090415820634), (95, 1.9720265943665387), (90, 2.6299556396765835), (93, 2.4776781245530843), (86, 4.546060565661952), (89, 4.163331998932265)]

[(86, 4.281744192888376), (89, 4.099457958749614), (98, 1.1547005383792515), (107, 0.9574271077563381), (94, 1.707825127659933), (91, 1.8633899812498245), (95, 1.2133516482134197)]

[(88, 2.266911751455907), (99, 1.4142135623730951)]

### NSMBO

[(111, 1.247219128924647), (83, 1.2909944487358056), (83, 1.2909944487358056)]

[(91, 4.618802153517006), (111, 1.3743685418725535), (97, 2.0), (95, 4.149966532662911)]

[(87, 4.5), (96, 2.494438257849294), (99, 0.9574271077563381), (95, 3.2871804872193366)]

[(106, 1.7950549357115013), (97, 1.8027756377319946), (86, 3.415650255319866), (89, 2.581988897471611)]

[(104, 1.674979270186815), (88, 3.1841621957571333), (87, 3.8042374035044424), (89, 2.4094720491334933), (85, 4.4969125210773475), (103, 1.8929694486000912)]

# P2

### MMOMBO

[(195, 7.425556469981821), (193, 7.610300037887249), (202, 2.309401076758503), (199, 3.8297084310253524), (198, 4.320493798938574), (204, 1.247219128924647), (205, 1.0)]

(205, 1.2909944487358056), (204, 1.8929694486000912), (192, 6.175669104549635), (206, 1.2472191289246473), (197, 2.211083193570267), (195, 2.4267032964268394), (194, 2.748737083745107)]

[(194, 9.546668994413123), (196, 9.370461864580399), (197, 2.7688746209726918), (195, 9.482439911049617), (201, 2.4776781245530843), (200, 2.6718699236468995), (211, 1.5)]

[(208, 1.950783318453271), (210, 1.2909944487358056), (241, 1.0671873729054748), (195, 3.144660377352201), (194, 3.337497399083464), (203, 3.0230595245361753)]

[(190, 9.086008780292673), (209, 1.0), (191, 4.654746681256314), (192, 3.2360813064912666), (194, 2.6299556396765835), (196, 2.217355782608345), (195, 2.3392781412697), (197, 2.1343747458109497), (199, 1.699673171197595)]

### NSMBO

# P3

### MMOMBO

### NSMBO

# P4

### MMOMBO

### NSMBO