基于遗传算法和贪心算法优化多元参数泊位分配问题

朱裕章

中山大学数据科学与计算机学院

摘要：针对简单三参数（到达时间、船体大小、需要的时间）的泊位分配问题，利用加以优化的遗传算法进行研究。在算法设计上，利用船只编号编码求解船只调度，辅以贪心算法提高港口泊位利用率，并将港口利用率反映在评估结果上。通过实验数据评测，证实在贪心算法解决船只调度问题的基础上利用船只编号编码一维基因的可行性、编程便捷性和有效性，以及经过优化设计的遗传算法能够有效避免求解过程的“早熟”现象和局部最优困境。

关键字：泊位分配；遗传算法；贪心算法；参数优化

引 言

在现如今航海技术的发展以及经济全球化带来的集装箱船运的市场需求，远洋运输不断发展。但是远洋运输业的带来了港口集聚群竞争的加剧，在给港口发展和港口商业市场带来机遇的同时，也给港口船只调度问题提出了很大的挑战。在这样的背景之下，对港口船舶调度问题的研究不断开展，泊位分配问题（Berth Allocation Problem，BAP）由此被提出。泊位分配问题是当今集装箱港口船舶服务及作业效率的瓶颈，考虑到船舶到达时间、吃水量（即船只大小）、港口作业需要的时间、优先度、约定离港时间、港口服务费用、超时罚款等等不定随机因素，泊位分配问题已经被证明为 NP 难完全问题。在近现代各种智能算法被发明出来的背景下，各项利用遗传算法、模拟退火、粒子群等启发式算法进行计算和分析，建立各种约束模型进行求解。由于本次实验针对的是简单三参数（到达时间、船体大小、需要的时间）的泊位分配问题，所以在目标函数方面，以最小化评估函数值为目标，以假设优先级相同的前提下，忽略其他因素的影响。

在本次实验中用到的遗传算法，是一类借鉴生物界的进化规律（适者生存，优胜劣汰遗传机制）演化而来的随机化搜索方法。它是由美国的 J.Holland 教授 1975 年首先提出，其主要特点是直接对结构对象进行操作，不存在求导和函数连续性的限定；具有内在的隐并行性和更好的全局寻优能力；采用概率化的寻优方法，能自动获取和指导优化的搜索空间，自适应地调整搜索方向，不需要确定的规则。遗传算法的这些性质，已被人们广泛地应用于组合优化、机器学习、信号处理、自适应控制和人工生命等领域。它是现代有关智能计算中的关键技术之一。根据问题实际情况，对应更改遗传算法的参数或者优化遗传算法的架构，能够高效地求得近似最优解。

1 问题描述

在一个集装箱码头上，像香港集装箱码头，泊位分配问题的瓶颈问题制约着码头的服务和作业效率。所以，码头需要对有限的船舶进行高效的泊位分配。

假设一个集装箱码头有n个泊位和 m 艘即将到达的船只。每艘船至少需要一个泊位用于装卸集装箱。船只 i（i = 1,2⋯,m）在a_i单位时间后到达，并且需要b_i个泊位和t_i单位时间才能完成装载货物。要求同一个泊位不能有不同的船只同时进行作业；分配的泊位作业时间段起始点在船只到达时间之后。在以上条件基础上对给定的港口以及船舶信息进行泊位分配。

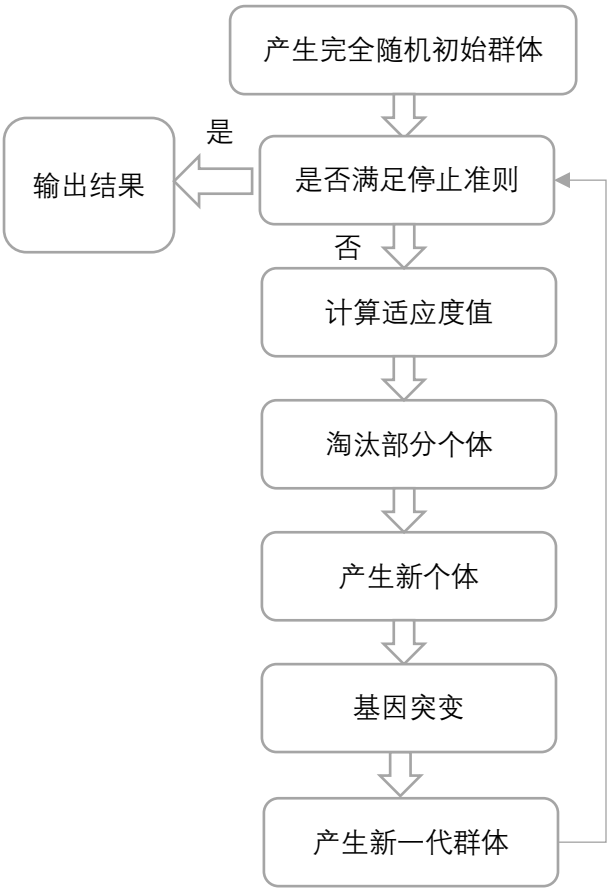
对于每一种泊位分配方案，未分配泊位的船只数权重$w_1 = 100$；所有船只总共等待时间（开始作业时间 – 到达时间）权重$w_2 = 2$；最后一艘船离港时间权重$w_3 = 1$。假设方案中未分配泊位的船只数为x_1；所有船只总共等待时间为x_2；最后离港时间为x_3，港口泊位分配方案效率评估函数：

$$\mathrm{f(x)} = w_1x_1 + w_2x_2 + w_3x_3 \quad (1)$$

2 算法规划

借鉴生物进化论，遗传算法将要解决的问题模拟成一个生物进化的过程。进化从完全随机个体的

种群开始，之后一代代发生，在整一代群体中，每个个体的适应度被评价（利用评估函数值），通过淘汰、交叉、突变等操作产生下一代的解，并逐步淘汰掉评估函数值高的解，增加评估函数值低的解。这样进化 N 代后就很有可能会进化出适应度函数值很低的个体。



2.1 主要参数说明

Group = $\{I_1, I_2, \dots I_N\}$：种群；
I_i：携带染色体x_i和评估函数值的个体；
Vessels = $\{v_1, v_2, \dots, v_m\}$：船只集合；
v_1：携带信息的船只；
Generation = 100 - 200：种群进化的迭代次数
Elite = 20%~50%：精英比例
Mortality = 20%：淘汰率
AberrationRate = 1%：变异率
TotalWaitingTime：总共等待时间
LastDepartureTime：最后离港时间
UnassignedVessel：为分配船只数

2.2 编码

由于需要将问题的解编码成字符串的形式才能使用遗传算法。而编码形成的字符串作为个体的染色体，包含着问题的解。由于本实验利用每一艘船优先安排顺序来作为问题的解，所以将每一艘船的编号作为每一位基因，编号序列作为染色体，编码形式为十进制整型数组。

2.3 适应度评估

利用问题描述中的方案效率评估函数值作为每一个个体的适应度值，需要注意的是，这里适应度值和个体的“生存能力”成反相关关系，即适应度值越大，个体的“存活能力”越小。

对给定的个体染色体（船只安排顺序），然后按染色体顺序进行泊位分配，利用公式（1）统计评估个体的适应度。

2.4 个体淘汰

遗传算法使用选择运算对个体进行优胜劣汰操作。评估值低的个体被遗传到下一代群体中的概率大；评估值高的个体，被遗传到下一代群体中的概率小。选择操作的任务就是从父代群体中选取一些个体，遗传到下一代群体。在本程序中利用基本的轮盘赌选择方法。

轮盘赌选择方法：设群体大小为n，个体x_i的评估值为$f(x_i)$，个体x_i的选择概率为

$$P(x_i) = \frac{f(x_i)}{\sum_j^N f(x_j)} \quad (2)$$

利用如下模拟方式实现：

（1） 在[0,1]之间产生一个随机数 r;

（2） 若$\mathrm{r} < q_i$，则个体x_i被选中。其中q_i为x_i的积累概率，计算公式为：

$$q_i = \sum_1^i P(x_i) \quad (3)$$

实现步骤：

（1）计算群体中所有个体的评估函数值；

（2）计算每个个体的选择概率；

（3）计算积累概率；

（4）采用模拟赌盘操作（即生成 0 到 1 之间的随机数与每个个体遗传到下一代群体的概率进行匹配）来确定各个个体是否遗传到下一代群体中。

（5）重复（4），知道淘汰个体数达到设定值。

2.5 产生新个体

由于每个个体携带的染色体为单体，即一维数组，所以采取交叉运算两个个体的染色体来产生新个体。为了尽量避免产生的子体和母体相似，提升基因多样性，这里采用顺序交叉(Order Crossover)的方法来产生新个体。

顺序交叉法：从父代 A 中随机截取一段染色体，放到子代 A 染色体中对应位置，子代 A 染色体

中其他空余部位按照父代 B 中的基因顺序进行选取（跳过已有基因）。同理可得子代 B。

父代 A：A B / C D E F / G H I

父代 B：h d / a e i c / f b g

子代 A：a i C D E F b g h

子代 B：B D a e i c F G H

产生新个体的流程：

（1） 随机从种群中挑出两个个体；

（2） 随机截取部分染色体片段；

（3） 利用顺序交叉法生成两个新个体；

（4） 将新个体加入新一代种群中；

（5） 重复（4）步骤直到产生足够的新个体；

上述产生随机挑选个体的方法不采用选择算子，即不使用轮盘赌或者锦标赛等选择方法选择父代，也就是说每一个个体都有均等的机会产生新个体。这不符合基本的遗传算法的基本规则，这个方法的使用原因会在后面解释。

2.6 基因突变

由于利用船只编号进行编码染色体，所以每一位基因都是唯一的，所以不能利用位反转或者位变换的方式进行基因突变运算。这里提出了一个类似交叉的基因突变方法：

（1） 遍历每一个新个体；

（2） 产生一个 0-1 的随机数 r；

（3） 当 r < *AberrationRate(变异率)*，随机挑选出两位基因进行交换；

（4） 重复（2）（3）直到新个体完全遍历。

2.7 程序伪代码

```
Procedures GA： 伪代码
begin:
    Initialize();//初始化第一代
    generation : 0;
    Generation://固定迭代次数
    while (generation < Generation)
    begin
        Fitness();//计算群体中每一个个体适应度

        Selection();//轮盘赌选择淘汰部分个体

        Cross();//交叉运算产生新个体

        Aberration();//对新生个体进行变异操作

        GenerateNewGroup();//将旧群体中部分优良个体和新生体组成新群体
        IfCatastrophe();//判断群体是否发生进化停滞情况，决定是否产生灾变
            generation++;
        ChangeParameter();//根据情况调整参数
        end
end
```

3 算法优化

针对问题的特定情况，对遗传算法进行适应性的优化。

利用贪心算法计算适应度：由于染色体为船只优先安排顺序并且泊位和总共服务时间在平面展开为二维结构。所以如果单纯利用排队方法（即在服务时间维度上先后分配泊位）计算评估函数值，那么会出现后到的船先分配而造成前面的服务时间和泊位浪费或者船体小的先分配造成同时间段泊位没有利用的不符合实际的情况。故引入贪心算法，即按照染色体中的编号顺序优先安排，从船只到达时间开始往后计算，遍历泊位，如果出现空泊位并且泊位数量足够，那么直接分配对应泊位直到船只分配到泊位或者没有足够泊位以及服务时间可以分配为止。实验数据表明，引入贪心算法能够大大减小每一个染色体的评估函数值。

根据种群大小选择是否淘汰个体：显然在种群进化的初期，种群群体比较小，而这个时候重复的染色体数很少，所以为了染色体的多样性，避免过早淘汰个体导致群体“早熟”，提前收敛到局部最优解，所以设置了一个特定的种群保护期，即在群体数量达到一定之前自然不做选择淘汰。

精英主义(Elitist Strategy)选择：当利用交叉和变异产生新的一代时，我们有很大的可能把在某个中间步骤中得到的最优解丢失或者在交叉和突变步骤中将染色体中某一段优良基因破坏掉。为了防止进化过程中产生的最优解被交叉和变异所破坏，在每一次产生新的一代时，首先把当前最优解原封不动的复制到新的一代中。精英主义方法可以大幅提高所求解的精度，因为它可以防止丢失掉找到的最好的解。但是同时带来的问题是群体不断扩大，影响运算速度。这也是为什么本程序在选择父代没有使用选择算子的原因，因为最优部分个体已经保留下来，剩下的主要问题是避免陷入局部最优解。

精英主义(Elitist Strategy)选择极端化：在计算每一个个体染色体的适应度的时候，其实也是在求解每一个方案的评估函数值，所以在每一次计算的

时候，都把最优个体保留下来，然后在群体进化的最后做最优解判断即可。这种方法能够保证出现的最优解不会被破坏或者淘汰掉。

灾变(Catastrophe)：由于遗传算法的局部搜索能力较强，但是很容易陷入局部极值。所以要跳出局部极值就必须杀死当前所有的优秀个体，从而让远离当前极值的点有充分的进化余地。引入灾变就是为了杀掉最优秀的个体，这样才可能产生更优秀的物种。当判断连续数代最佳染色体没有任何变化时，或者各个染色体已过于近似时，即可实施灾变。灾变方法很多，可以突然增大变异概率或对不同个体实施不同规模的突变。本程序实施大规模灾变，即将老群体完全杀死。这个方法看似和精英主义矛盾，两者其实是可以共存的。我们在每一代进行形成新种群的时候时,均直接把最优秀的个体复制到下一代;但当连续 N 代,都没有更优秀的个体出现时,便可以猜想可能陷入局部最优解了,这样可以采用灾变的手段.可以说,精英主义是伴随的每一代的,但灾变却不需要经常发生,否则算法可能下降为随机搜索了。

动态确定变异概率、淘汰概率、精英比例：在种群发展的初期，种群比较小，为了保证染色体的多样性和避免陷入局部最优解和“早熟”现象，淘汰的概率应该比较低，而精英比例应该稍微高一点。到了种群发展的中后期，出现了大量的相似或者相同的个体，这时候种群发展缓慢，而且相似或者相同的个体占据种群大部分比例，所以适当地提高淘汰概率和变异概率，既能够把评估值高的个体淘汰掉，把近似个体剔除，又能为群体引入较新的染色体，提高算法效率以及避免局部最优解。同样，种群量在中后期比较大，如果固定精英比例，那么近似精英数量越来越多，占据了种群大部分空间，也可能导致群体陷入局部最优，所以适当地下调精英比例，为新生个体腾出空间，同时也能提高算法运算速率。

4 结果分析

4.1 程序运行速度

首先来看一些程序的参数设置和运行速度：

表 1：程序参数设置和运行速度

数据存储	淘汰率(%)	精英比例(%)	变异率(%)	群体保护量	初始量	迭代数	运行时间(s)
指针存取	20	33.33	1	50	10	100	15
非指针存取	20	33.33	1	50	10	100	30-40
非指针存取	20	25	1	50	10	100	0.5-1
非指针存取	40	50	1	50	10	100	0.3-0.5
非指针存取	30	50	5	70	10	100	10
非指针存取	40	50	1	70	10	150	0.5-1
非指针存取	40	50	1	500	50	200	1-5
非指针存取	20	20	10	500	50	200	5
非指针存取	引入灾变，动态变换上面三个量			100	50	250	1

注：群体保护量指群体量没达到设定值不做淘汰；运行时间是粗略值

从上图可以看出，程序运行时间并不是按照线性变化的，二是按照指数形式增长。如果淘汰率越低，精英比例越高，迭代次数越多，那么程序运行越加耗时。那么，可以得出结论，程序运行速度主要取决于群体增长速度和约束特性，群体量大而约束特性小则处理计算的数据多，不过考虑到群体量大，数据多，更容易求得近似最优解。

4.2 算法对比

表 2：贪心算法和遗传算法对比

	贪心算法				遗传算法（最优解）			
游戏	未分配船	等待时间	离开时间	评估值	未分配船	等待时间	离开时间	评估值
1	1	5	5	115	0	8	6	22
2	1	0	6	106	0	0	7	7
3	0	19	17	55	0	19	17	55
4	1	33	19	185	0	30	20	80
5	1	4	9	117	0	4	9	17
6	1	7	10	124	0	7	10	24
7	1	26	10	162	0	25	10	60
8	1	33	16	182	0	38	16	92
9	1	47	19	213	0	63	20	146
10	2	93	20	306	0	59	19	137
11	2	98	24	420	0	97	24	218
12	2	61	29	351	0	50	30	130

使用遗传算法，能够给每一艘船都分配到泊位和服务时间，经过特定优化和设计的遗传算法则具有较高的搜索能力和极强的鲁棒性，而且在数据量越大的情况下这种优势越明显。不过可以发现，在数据量比较少的时候，贪心算法具有时间上的优势，而且更利于减少等待时间，所以，基于贪心算法的遗传算法，在发挥了贪心算法在追求各项值最小的优势的前提下，避免了贪心算法的局部最优困境。

4.3 算法性能分析

本程序测试了十二个样例，为了体现出遗传算法的鲁棒性，现选取数据量较大的样例 11 来进行评估算法性能。

表 3：样例 11

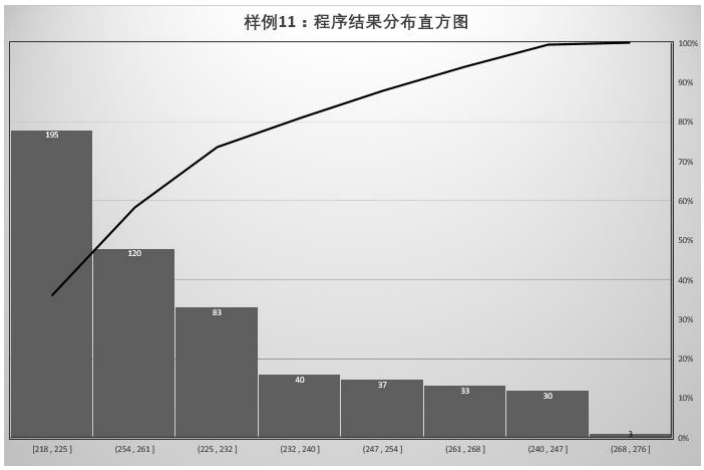
	服务总时长	港口泊位数	船舶数量
	24	8	20
编号	到达时间	时间需求	泊位需求
1	0	5	5
2	0	4	1
3	0	5	4
4	1	2	4
5	1	5	5
6	2	5	3
7	2	1	2
8	2	3	4
9	3	3	5
10	3	6	2
11	3	1	1
12	3	3	5
13	3	1	1
14	4	5	1
15	4	1	2
16	6	2	1
17	7	2	4
18	7	1	5
19	8	1	3
20	8	1	1

在程序中读入运行样例 11 的数据之后，获取到 541 份运行数据，每一份数据的运行结果视参数设置而定。然后整理分析，得到了样例 11 近似最优解为$f(x) = 218$，占比 11.65%。如表 4 及图 1：

表 4：运行结果频率

接收区间	频率	累积 %
0	0	0.00%
218	63	11.65%
220	58	22.37%
230	136	47.50%
240	67	59.89%
250	38	66.91%
260	129	90.76%
270	49	99.82%
280	1	100.00%
其他	0	100.00%

图 1：程序结果分布直方图



需要说明的是，近似最优解 218 中的大部分都是在非指针操作，淘汰率 20%，精英比例 33.33%，变异率 1%，保护量 50，迭代 100 代，初始量 10 的参数设置之下求得的，而引入灾变和动态确定概率和精英比例之后，近似最优解的比例不断上涨。

从上表可以评估本程序的性能，在运行结果评估值 230 以下的概率已经很接近 50%，评估值高于 260 的运行结果仅仅占比 10%。大部分运行结果都趋近于 218。

表 5：样例 11 结果数据分析

平均	237.5748614
标准误差	0.721417355
中位数	232
众数	218
标准差	16.77973966
方差	281.5596632
峰度	-1.516304249
偏度	0.319191711
区域	54
最小值	218
最大值	272
观测数	541

在大量求得的数据结果当中，众数为求得的近似最优解，中位数 232 和近似最优解 218 相差也不大。从上面表格中各项数据看来，本程序的算法性能优良，能够快速地对本问题求出近似最优解。

5 遗传算法优缺点

遗传算法具有较高的搜索能力和极强的鲁棒性，但是适应度值标定方式多种多样，没有一个简洁、通用的方法，不利于对遗传算法的使用。遗传算法的早熟现象(即很快收敛到局部最优解而不是全局最优解)是迄今为止最难处理的关键问题。在实现本程序的过程也遇到了“早熟”现象，程序运行结果集中在某个局部最优解区段。这说明了在程序运行早期，较优个体迅速占领了种群的大部分空间，迭代求解一直在局部最优解附近摇摆。所以在优化程序的时候，引入个灾变和动态调整参数，让程序能够及时跳出局部最优解。

而且遗传算法全局搜索能力不强,很容易陷入局部最优解跳不出来。将遗传算法用于解决各种实际问题后，人们发现遗传算法也会由于各种原因过早向目标函数的局部最优解收敛，从而很难找到全局最优解。其中有些是由于目标函数的特性造成的，例如函数具有欺骗性，不满足构造模块假说等等；另外一些则是由于算法设计不当。为此，不断有人对遗传算法提出各种各样的改进方案。例如：针对原先的定长二进制编码方案；提出了动态编码、实数编码等改进方案；针对按比例的选择机制，提出了竞争选择、按续挑选等改进方案；针对原先的一点交算子，提出了两点交、多点交、均匀交等算

子；针对原先遗传算法各控制参数在进化过程中不变的情况，提出了退化遗传算法、自适应遗传算法等。另外，针对不同问题还出现了分布式遗传算法、并行遗传算法等等。

6 总 结

有研究证明，简单遗传算法在任何情况下（交叉概率，变异概率，任意初始化，任意交叉算子，任意适应度函数）都是不收敛的，且不能搜索到全局最优解。

所以为了提升遗传算法的搜索能力，在参数不多而且不考虑船舶调度优先度的情况下，利用贪心算法为船舶调度设置优先度，有利于降低每一个调度方案的效率评估值。在遗传算法中的适应度评估中引入贪心算法，利用贪心算法在追求各项值最小的前提下，遗传算法作为主体搜索每一种优良方案，并进行比较和取舍，避免了贪心算法的局部最优困境，同时提升算法的性能，即提高搜索出最优解的概率

在针对特定的情况，对遗传算法引入灾变、精英主义等跳出局部最优解的手段，可以大幅提高运算速度和搜索全局最优解的精准度。但是，引入这些手段需要测试各项参数之间的相互影响关系，找到协同提升算法性能的参数设置集，才能将遗传算法的搜索能力发挥得更好。在本程序中，虽然经过不断尝试和测试，针对某些样例仍然不能很好地求出近似最优解，“早熟”现象时有发生，算法效果并没有预期那么理想，但是如果找到适应该问题的参数设置集，相信该程序算法能够很好地对该问题进行求解。

Reference

[1]. 《经典算法研究系列：七、深入浅出遗传算法》,
http://blog.csdn.net/v_JULY_v/article/details/6132775
[2].《遗传算法小结》;
http://blog.csdn.net/tsroad/article/details/52448313
[3].《遗传算法 GA(Genetic Algorithm)入门知识梳理》;
https://segmentfault.com/a/1190000004155021