利用模拟退火算法解决港口停泊位分配问题

学生姓名：张镓伟

学　　号：15352408

学　　院：数据科学与计算机学院

专　　级：软件工程(移动信息工程)

年级班别：1518

邮　　箱：709075442@qq.com

指导老师：张子臻

2017 年 6 月 1 日

摘要：

在一些集装箱码头，我们总是能遇到因为停泊位不够而造成大量的船在等待，从而浪费了大量时间的问题。本文讨论如何使用模拟退火算法（SA）去解决港口停泊位分配问题(Berth Allocation Problem)，力求在规定时间内能安排尽量多的船停靠并完成所需服务，而且各船的等待时间之和最少。本文还会通过对同样数据的测试，分析比较模拟退火算法和贪心算法得出的结果。

关键词：港口停泊位分配问题、模拟退火算法、贪心算法。

目录

1 问题介绍

1.1 问题背景

在现实生活中，一个港口总是需要容纳很多在不同时间到来的船只并提供相应的服务。船只在港口指定位置停靠后，就要开始装卸物资、加油、打扫清洁等，而这些行为需要一定的时间去完成，我们暂且将这个时间称为服务时间。

一个港口的停泊位总是有限的，我们总是不可避免的遇到这种情况：当一艘船到达港口时，所有停泊位都已经停满了船，那么它只能选择等待直到有船离开再开始停靠。

由于每艘船所需的服务时间是不同的，作为一个港口的管理者，我们希望找到一种停靠方案，使得在尽量短的时间内完成对尽量多的船的服务，从而提高港口的吞吐量。

上文需要解决的问题就是港口停泊位分配问题（Berth Allocation Problem），简称 BAP。它是一个 NP 问题。在数据量较小的情况下，我们可以选择爆搜得到解；但是在数据量较大的情况下，爆搜耗费的时间是巨大的，是我们所不能接受的。因此目前比较有用的方法是使用现代优化算法去求一个近似解。在接下来的文章中我会先分析说明使用贪心和暴力搜索求解特点，再介绍如何使用现代优化算法中的模拟退火算法（SA）求解 BAP。

1.2 问题的数学描述

基于老师给出的网站游戏上的数据格式，我们不妨设一个港口有 L 个连续的停靠位(0<L<=12)，编号为 0、1、…、L-1,它希望求解在 T(0<T<=30)时间内的服务方案。有 n 艘船要到达该港口，每艘船有三个属性 ai、bi、ti，(1<=i<=n)，其中 ai 表示第 i 艘船的到达时间 ai（0<=ai<T）,bi 表示第 i 艘船需要占用 bi 个连续的停泊位，ti 表示第 i 艘船的服务时间。我们需要求解一个停靠方案，使得函数 f(S) (S 的定义见下文)的值最小，其中

f(S) = 100*x1 + 2*x2 + 3*x3

x1: 未能完成停靠的船数，一艘船完成停靠是指它在 T 时间内成功停靠并完成相应的服务而离开，即最晚离开时间为 T。若在 T 时刻未能完成服务也视为未完成停靠。

x2: 总等待时间，即每艘完成停靠的船开始停靠的时间与其到达港口的时间之差的总和。

x3: 完成停靠的船中最后一艘离开的船的离开时间

我们将完成停靠的船的编号放入集合 S，称为解空间，对解空间中的每艘船 i，设其开始停靠时间为 ti’,开始停靠位置为 bi’，将停泊位与时间组成一个 L*T 的二维矩阵 G，那么一个船完成停靠可以看它占了其中的一个小矩形 Ci={bi’<=b<=bi’+bi-1, ti’<=t<=ti’+ti-1, $i \in S$ },则上述问题可以表述成下列形式：

$$\min_{x1,x2,x3} f(S) = 100 * x1 + 2 * x2 + 3 * x3$$

$$\text{s.t.} \quad x1 = n - |S|$$

$$x2 = \sum_{i \in S} ti' - ai$$

$$x3 = \max(ti' + ti) \qquad i \in S$$

$$ti' + ti \le T \qquad i \in S$$

$$bi' + bi \le L \qquad i \in S$$

$$\bigcap_{i \in S} Ci = \emptyset \qquad Ci \in G \;\; G = \{0 \le b < L, 0 \le t < T\}$$

$$Ci = \{bi' <= b <= bi' + bi - 1, ti' <= t <= ti' + ti - 1, i \in S\}$$

2 传统算法

2.1 贪心算法

2.1.1 贪心算法描述

贪心算法是指在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。

在本问题中，我们可以采取如下贪心策略：按照船的到达顺序询问，假设当前船到达的时候有位置可停，则停，而且停靠的起始位置尽量小，即取一个最小的可行的 bi’ 。如果没有位置可停，则一直等待直到有船离开再停。

将上述贪心策略按照 1.2 中的数学模型转换一下，每个船的停靠看成是将一个 bi*ti 的小矩形填入到G这个大矩形中，并且矩形尽量地往左上角填(优先让 ti’ 尽可能小，再让bi’ 尽可能小)且满足 ti>=ai。 如下图，已经停好了一号船，现在停 2 号船，它可以在时间 3 之后从任意位置开始停，但根据我们的贪心策略，我们应该选择左上的位置，即图中红框处。

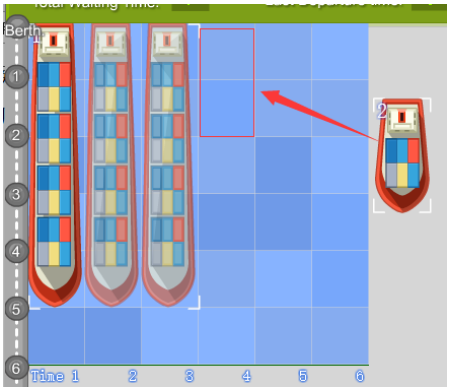

图 2.1.1.1

2.1.2 贪心算法局限性

前面我们说过，贪心只是考虑局部上的最优解，而不从整体上去考虑，也就是说，局部上的最优不一定导致全局的最优。按照我们的贪心策略，我们是按照船到达的顺序安排停靠，如果有两艘船到达时间一样，那么就先安排编号小的船，那么我们可能会遇到如下情况：按照贪心策略进行的 game 1 到最后 5 号船没有办法

图 2.1.2.1

然而实际上，如果我们在 1 号船停完后，先对 5 号船进行贪心分配位置则有办法停完。如下图：

图 2.1.2.2

对比两种情况，我们可以发现，两者仅仅是在停靠优先顺序上发生了变化，5 号船没办法停靠的安排顺序是 1、2、3、4、5；而全部船都能停靠时的安排顺序是 1、5、2、3、4。由此我们可以看出，安排停靠的优先顺序对贪心的结果起到了决定性的影响。

2.2 暴力搜索算法

2.2.1 暴力搜索算法描述

前面通过对贪心算法的分析，我们知道了，安排停靠的优先顺序对结果其决定性的作用。那么这就启示了我们得出一个暴力搜索算法：

首先先使用深度搜索得出关于 n 的全排列，对于每一个排列，再使用贪心算法去分配停泊位，从中找到最优方案即可。

2.2.2 暴力搜索算法局限性

由于暴力搜索算法考虑了所有的情况，所以这个算法是一定可以得到最优解的。但是我们知道，这个算法是建立在求 n 的全排列的基础上的，而在老师提供的数据中，n 最大为 24，$24! = 6.2044840173324 * 10^{23}$，如果按照计算机 1 秒钟运行 1 亿次去计算，单单搜索出 24 的全排列就要耗时约 19674289.76 年，那么在有生之年都不可能得到答案了，更不用说对每个排列还要求如何放。

综上所述，我们迫切地希望找到一种办法，将搜索次数缩小到我们可以接受的范围，但得到的结果能够尽量好。而这个办法就是接下来要介绍的模拟退火算法。

3 模拟退火退火算法

3.1 爬山算法(Hill Climbing)

在介绍模拟退火算法前，需要先简单介绍一下爬山算法。爬山算法是一种简单的贪心搜索算法，该算法每次从当前解的临近解空间中选择一个最优解替代当前解，直到得到一个局部最优解。

爬山算法的实现很简单，其主要缺点是会陷入局部最优解，而不一定能搜索到全局最优解，如图 3.1.1 所示，假设 C 点为当前解，爬山算法搜索到 A 点这个局部最优解就会停止搜索，因为在 A 点无论向那个方向小幅度移动都不能得到更优的解。

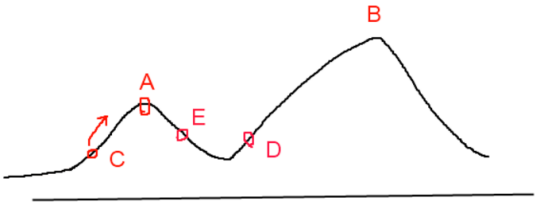

图 3.1.1

3.2 模拟退火算法介绍(Simulated Annealing)

模拟退火算法得益于材料统计力学的研究成果。统计力学表明材料中例子的不同结构对应于例子的不同能量水平。在高温条件下，粒子的能量较高，可以自由运动和重新排列。在低温条件下，粒子能量较低。如果从高温开始，非常缓慢地降温（这个过程被称为退火），粒子就可以在每个温度下达到热平衡。当系统完全被冷却时，最终形成处于低能状态的晶体。

如果用粒子的能量定义材料的状态，Metropolis 算法用一个简单的数学模型描述了退火过程。假设材料在状态 i 之下的能量为 E(i)，那么材料在温度 T 时从状态 i 进入状态 j 就遵循如下规律：

（1）如果 E(j)<=E(i)，则接受该状态被转换。

（2）如果 E(j) > E(i)，则状态转换以如下概率被接受：

$$e^{\frac{E(i)-E(j)}{KT}}$$

式中：K 为物理学中的玻尔兹曼常数；T 为材料温度。

当温度降至很低时，材料会以很大概率进入最小能量状态。

基于上述准则，我们考虑这样一个组合优化问题：优化函数为 $f:x \to R^+$，其中 $x \in S$，它表示优化问题的一个可行解，$R^+ = \{y | y \in R, y \geq 0\}$，S 表示函数的定义域。$N(x) \subseteq S$表示 x 的一个邻域集合。目标是求出一个 x，使 f(x)尽可能小。

首先给定一个初始温度T_0和该优化问题的一个初始解 x(0)，并由 x(0)生成下一个解$x' \in N[x(0)]$，是否接受x'作为一个新解 x(1)依赖于如下概率

$$P(x(0) \to x') = \begin{cases} 1 & , f(x') < f(x(0)) \\ e^{-\frac{f(x')-f(x(0))}{T_0}} & , others \end{cases} \quad (3.1)$$

换句话说，如果生成的解x'的函数值比前一个解的函数值更小，则接受 x(1)=x'作为一个新解。否则就医概率$e^{-\frac{f(x')-f(x(0))}{T_0}}$接受$x'$作为一个新解。

泛泛地说，对于某一个温度T_i和该优化问题的一个解 x(k)，可以生成x'。接受x'作为下一个新解 x(k+1)的概率为

$$P(x(k) \to x') = \begin{cases} 1 & , f(x') < f(x(k)) \\ e^{-\frac{f(x')-f(x(k))}{T_i}} & , others \end{cases} \quad (3.2)$$

在温度T_i下，经过很多次转移后，降低温度T_i，得到$T_{i+1} < T_i$。在T_{i+1}下重复上述过程。因此整个优化过程就是不断寻找新解和缓慢降温的交替过程。最终的解就是对该问题寻优的结果。

注意到在每个T_i下，所得到的一个新状态 x(k+1)完全依赖于前一个状态 x(k)，和前面的状态 x(0),…,x(k-1)无关，因此是一个马尔可父过程。使用马尔可父过程对上述模拟退火的步骤进行分析，结果表明从任何一个状态 x(k)生成x'的概率，在 N[x(k)]中是均匀分布的，且新状态x'被接受的概率满足式（3.2），那么经过有限次的转换，在温度T_i下的平衡态x_i的分布有下式给出:

$$P_i(T_i) = \frac{e^{-\frac{f(x_i)}{T_i}}}{\sum_{j \in S} e^{-\frac{f(x_j)}{T_i}}} \quad (3.3)$$

当温度 T 降为 0 时，x_i的分布为

$$P_i^* = \begin{cases} \frac{1}{|S_{min}|}, & x_i \epsilon S_{min} \\ 0 & , others \end{cases} \quad (3.4)$$

并且

$$\sum_{x_i \in S_{min}} P_i^* = 1$$

式中：$S_{min} = \{x_i | f(x_i) = \min_{x_j} f(x_j)\}$

这说明如果温度下降十分缓慢，而在每个温度都有足够多次的状态转移，使之在每一个温度下达到热平衡，则全局最优解将以概率 1 被找到。因此可以说模拟退火算法可以找到全局最优解。

在模拟退火算法中应注意以下问题：

(1)理论上，降温过程要足够缓慢，要使得在每一温度下达到热平衡。在计算机视线中，如果降温速度过缓，所得到的解会较好，但算法会太慢。而如果降温速度过快，则很可能最得不到全局最优解。因此使用时要综合考虑性能和算法速度。

(2)要确定在每一温度下状态转换的结束准则。比如确定一个较小的值作为终止温度，或者连续几次转换都没有发生状态变化则提前结束。

(3)选择初始温度和确定某个可行解的领域方法也要恰当。

模拟退火算法与爬山算法相比，爬山算法每次都鼠目寸光选择一个当前最优解，因此只能搜索到一个局部最优值，而模拟退火本质上也是一种贪心，但是它的搜索过程中引用了随机因素，以一定的概率接受比当前解要差的解，因此有可能会跳出这个局部最优解，达到全

局最优解。

下面以寻找一个最小值的优化问题给出一段模拟退火算法的伪代码：

```
/*
*F(STA)：在状态STA时的评价函数值
* STA(i)：表示当前状态
* STA(i+1)：表示新的状态
* Dec：  降温系数
* T：  系统的温度，系统初始应该要处于一个高温的状态，一般设为1
* T_min ：温度的下限，若温度T达到T_min，则停止搜索
*/
T=1; T_min=1e-30;
while( T > T_min ){
    dE = F( STA(i+1) ) - F( STA(i) ) ;
    if ( dE <=0 ) //表达移动后得到更优解，则总是接受移动
        STA(i+1) = STA(i) ; //接受从STA(i)到STA(i+1)的移动
    else{
        // 以exp(-dE/T)的概率接受 从STA(i)到STA(i+1)的移动
        if ( exp( -dE/T ) > random( 0 , 1 ) )
            STA(i+1) = STA(i) ;
    }
    T *= Dec; //降温退火 ， 0<Dec<1 。Dec越大，降温越慢；Dec越小，降温越快
    /*
    * 若Dec过大，则搜索到全局最优解的可能会较高，但搜索的过程也就较长。若Dec过小
      则搜索的过程会很快，但最终可能会达到一个局部最优值
    */
    i ++ ;
}
```

3.3 模拟退火算法在 BAP 问题中的应用

接下来我们回到需要解决的港口停泊位分配问题。在第 2 节使用传统算法解题时我们得出了船只停靠的优先顺序是解决这题的关键，也就是说我们需要搜索出一个合适的优先顺序。因此这里我们定义搜索状态就是 n 的一个排列。

模拟退火求解 BAP 问题的算法描述如下：

（1）解空间。解空间 S 可表示为{1,2,…,n}的所有排列。每一个排列表示的是船只的优先停靠顺序。比如当 n=5,解为{1,3,4,5,2}时，表示我们先考虑船 1 停靠，再考虑船 3 停靠，…,最后考虑船 2 停靠。而停靠的方法就是 2.1 节中介绍的贪心算法。初始解我们可以选择{1,2,3,4,5,…,n},并先用贪心生成一个停靠方案。

（2）目标函数。目标函数就是 1.2 节中定义的函数 f(S).要求。

$$\min_{x1,x2,x3} f(S) = 100 * x1 + 2 * x2 + 3 * x3$$

x1，x2，x3 的含义见 1.2 节。

每一次迭代由下面三步构成：

（3）新解的产生。设上一步迭代的解为 S(k)={$x_1, x_2, x_3, \cdots, x_n$}，有下面三种方法可以产生新的解。

A.任选序号 i 和 j，交换排列中的第 i 个数和第 j 个数，此时可得

S(k+1)={$x_1, \cdots, x_{i-1}, x_j, x_{i+1}, \cdots, x_{j-1}, x_i, x_{j+1}, \cdots, x_n$}

B.任选序号 i 和 j，交换 i 与 j 之间的顺序，此时可得

S(k+1)={$x_1, \cdots, x_{i-1}, x_j, x_{j-1}, \cdots, x_{i+1}, x_i, x_{j+1}, \cdots, x_n$}

C.任选序号 i,j,w，将 i 和 j 之间的数插入到 w 之后，此时可得

S(k+1)={$x_1, \cdots, x_{i-1}, x_{j+1}, \cdots, x_w, x_i, \cdots, x_j, x_{w+1}, \cdots, x_n$}

后两种方法更多地适用于数据较大的情况，此次的数据中船只数量 n 最大只有 24，不必使用后两种方法，所以我只使用了方法 A 去产生新解。产生新解后使用贪心算法对新解求出对应的目标函数。

另外有一点，我为了能够提高求得最优解的概率，我同时保存了 7 个最优解，每一次迭代，7 个解都各自产生一个新解去更新自己。

（4）目标函数差

df=f(S(k+1))- f(S(k))

(5) 接受准则

$$P=\begin{cases}1 & ,\ df<0\\ \exp(-df/T) & ,\ df\geq 0\end{cases}$$

如果 df<0 则接受新解，否则以概率 exp(-df/T)接受新解，即使用计算机产生一个[0,1]区间上均匀分布的随机数 rand，若 rand<=exp(-df/T)则接受。

(6)降温。利用选定的降温系数 Dec=0.999 进行降温，取新的温度 T’=Dec*T（T 为上一步的温度）。初始温度我设为 T_0=1.

(7)结束条件。选定终止温度 $e=10^{-30}$，若 T<e 则结束算法。

算法伪代码如下：

```
Simulate_Anneal ()
    choose an initial solution S0 by greedy algorithm
    give an initial temperature T0←1,T←T0,Dec=0.999
    give an end temperature     e←1e-30
    initialize the best 7 solution by S0    p[i] ←S0，0<=i<=6
    initialize the ANS by S0     ANS←S0
    while T>e
        for i=0 to 6
            pick a solution temp∈N[P[i]] randomly
            use greedy algorithm to get f(temp)
            df ←f(temp) - f(P[i])
            use temp to update ANS (if temp is better than ANS)
            if df <0 then
                p[i] ←temp
            else if exp(-df/T)>rand then
                p[i] ←temp
        end
        T*=Dec
    end
    return ANS
```

完整代码详见 code 文件夹

4 实验结果

本次实验前 12 个数据均来自老师提供的网站上 12 个游戏的数据，第 13 个数据是我随机生成的数据，是 T=30，L=12，n=24 的大数据，具体见 data 文件夹中的 13.in。下面是使用贪心算法得出的结果和使用模拟退火算法得出的结果比较。为了更清晰地对比两种方法的结果，下面只展示出两种方法得出的 Unassigned vessels (x1) ，Total waiting times (x2)，Last departure time (x3)和 f(x)=w1x1+w2x2+w3x3。至于停靠的方案，详见 data 文件夹中对应的 out 文件

	Unassigned vessels (x1)		Total waiting times (x2)		Last departure time (x3)		f(x)=w1x1+w2x2+w3x3	
GAME	贪心	模拟退火	贪心	模拟退火	贪心	模拟退火	贪心	模拟退火
1	1	0	5	8	5	6	115	22
2	1	0	0	0	6	7	106	7
3	0	0	19	19	17	17	55	55
4	1	0	33	30	19	20	185	80
5	1	0	4	4	9	9	117	17
6	1	0	7	7	10	10	124	24
7	1	0	26	25	10	10	162	60
8	1	0	33	38	16	16	182	92
9	1	0	47	63	19	20	213	146
10	1	0	93	59	20	19	306	137
11	2	0	98	97	24	24	420	218
12	2	0	61	46	29	30	351	122
13	7	3	45	30	30	30	888	420

从上表可以看出，使用了模拟退火算法之后，结果要比使用只贪心算法得出的结果要优秀很多。这也从说明了模拟退火算法在求 NP-hard 组合优化问题的全局最优解中有着很好的优化作用。

若要使用我的程序进行实验，可以先查看根目录下的 readme

5 总结

现代优化算法是 20 世纪 80 年代初兴起的启发式算法。模拟退火就是其中之一。它主要用于解决大量的实际应用问题，目标是求 NP-hard 组合优化问题的全局最优解，例如本文中的港口停泊位分配问题，取得了良好的效果。善用模拟退火算法，能帮助我们解决复杂优化问题，当然也要注意算法中初始温度，降温因子的选择，以及结束准则和确定可行解领域的方法，这几点是决定模拟退火算法成败的重要因素。

6.参考读物：

[1]司守奎 孙兆亮
《数学建模算法与应用（第 2 版）》（2015）
[2]Thomas H. Cormen Charles E.Leiserson Ronald L. Rivest Clifford Stein
《Introduction to Algorithms ， Third Edition》 (2013)