模拟退火在港口停船分配问题上的应用

学　　生：梁杰鑫

学　　号：15352514

学　　校：中山大学

学　　院：数据科学与计算机学院

专　　业：软件工程（移动信息工程）

年级班别：1509

指导老师：张子臻

2017 年 6 月 3 日

摘要

在船只来往密集的码头中，我们常常会遇到因为港口停泊位置有限而让其余船只等待的情况。如果船只分配不好，就会浪费大量的时间。如何分配船只是一个 NP-HARD 问题，本文尝试基于贪心算法，利用模拟退火来求解停泊问题的较优解，并和一般的贪心算法和 DFS（深度优先搜索）进行比较。择优条件为船只总等待时间和船只总服务时间（最后一只船服务完毕的时间）越小越好。

关键词

港口停船分配问题，模拟退火算法，NP-HARD 问题

目录

引言

在现实生活中，一个港口会容纳很多在不同时刻进入港口，大小不一的船只。船只会被安放在港口的指定位置，然后开始装卸货物。在管理港口的时候，除了调动各种设备和人力资源去让物流运转之外，我们还要增加港口的利用率和吞吐量，在花费尽量少的资源和时间的情况下，服务更多的船只。

本文讨论的主要问题是讨论如何将这些船只安排到港口中合理的位置，并在合适的时间点开始船只服务。所以我们求解的输出有两个：船只进港的位置，船只进港的时间。考虑现实生活，这个问题还有若干个约束条件：进港时间不能冲突，进港的船只数不能超过港口的承载能力，港口能够服务船只的总时间等等。

这个问题按照传统的搜索方法，我们一般会采用 DFS（深度优先搜索）和 BFS（广度优先搜索来得到）这个问题的最优解决方案。但是这是一个 NP-HARD 问题，只有在数据量比较小的时候才能用遍历搜索；在数据量大的时候，我们可能穷尽一生也不可能搜索完毕。因此现代算法对这种问题提出了“求近似解”的思路，本文主要探讨的模拟退火算法就是其中一种。

1. 问题介绍

本文的解题方案基于指导老师所给出的港口停泊问题（Berth Allocation Problem，下面简称 BAP）模拟器来讨论算法的。

对于这个问题，我们采用了这样的建模：首先建立一个直角坐标系；停泊位置是 Y 轴，船只的服务开始时间是 X 轴。这样一来，每一艘船只可以看做是直角坐标系中的一个矩形；对于船只，我们设定了 3 个主要参数，到达时间，服务时间，船只长度。

$V = \{v_0, \dots v_{N-1}\}$：船只集合。

$v = (t_{arrive}, t_{service}, berth)$：船只的构成。

于是我们便得到了如下图的模型。

我们的目标就是利用这个模型来求解 BAP 问题的近似最优方案。对于每一个解，我们定义了 5 个成员：

$S(V) = (B, T, U, W, D)$：解的构成。

$B = \{b_0, \dots b_{N-1}\}$：船只占据港口的起点。

$T = \{t_0, \dots, t_{N-1}\}$：船只开始服务的时间。

$U = count(b_i == null)$：没有被分配的船只总数。

$W = \sum_{i=0}^{N-1} v_i(t_{arrive}) - t_i$；船只的总等待时间。

$D = max\{t_i + v_i(t_{service})\}$：船只的总服务时间。

为了辅助解题我们在定义一个数据结构

$M(S, V)_{Berth*Time}$：船只摆放的矩阵。

对于每个解的评估函数，我们有：

$$f(S) = 100 * S(U) + 2 * S(W) + S(D)$$

约束条件为：

$$W \leq Time \quad (1)$$

$$max\{b_i + v_i(berth)\} \leq Berth \quad (2)$$

$$\forall t \in T, t_i \geq v_i(t_{sevice}) \quad (3)$$

$$\forall v_i \in V, \forall v_j \in V, i \neq j,$$
$$C_i\{M[p][q] | b_i \leq p < b_i + v_i(berth), t_i \leq q \leq t_i + v_i(t_{service})\}$$
$$\cap\, C_j\{M[x][y] | b_j \leq x < b_j + v_j(berth), t_j \leq y \leq t_j + v_j(t_{service})\} = \emptyset \quad (4)$$

在模型中我们设置了约束条件：1、船只摆放的总长度不能超过 X 轴限制；2、船只摆放总宽度不能超过 Y 轴限制；3、在时间小于船只到达时间时，船只不能被服务；4、船只不能重叠。

这些条件使得这个问题变成了一个矩形的装箱问题。

2. 一般算法

2.1. 贪心算法

2.1.1.具体方案

贪心算法是在对问题求解时，做出目前看来最佳的选择，不考虑未来和旁路的可能性。也就是说，贪心算法是不会考虑整体最优的。

根据评估方法，b_i并不在评估的范围之内，所以我们最先保证的就是t_i小。于是，我们的贪心策略是：选一个能够停船的位置，先保证t_i尽可能小，再保证b_i尽可能小。

具体的做法就是按照先到先服务的原则，顺序访问船只队列V，按照前面所说的贪心原则摆放船只就行。

2.1.2.局限性

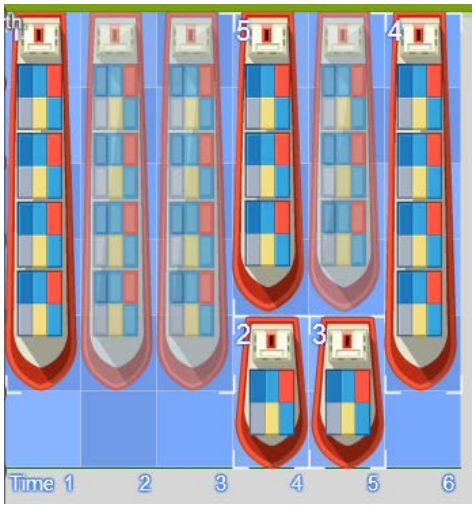
我们所说的贪心算法只会考虑局部最优解，也就是说，他不会考虑按照其他顺序访问船只队列，去寻求更优的解。

以 Game1 为例子，贪心算法的访问顺序是 1-2-3-4-5

Time Table of Vessel Arrivals

Vessel index	Arrival time	Service Time	Berths occupied	Icon
1	0	3	5	
2	1	1	2	
3	2	1	2	
4	2	1	5	
5	2	2	4	

此时贪心算法得到的解就不能放下所有船只。但是我们只需要交换一下访问顺序：1-5-2-3-4，就可以得到一个可以摆放所有船只的解。（下页）



可以看出，如果要得到全局最优解，我们必须穷尽所有船只访问顺序，这会放到下一节讲。

2.2. DFS 穷举

2.2.1. 具体方案

根据前面贪心算法的局限性分析，我们知道我们需要穷举所有船只的访问顺序。本质上讲，我们需要求出船只队列的全排列，对与每一种排列都采用贪心算法求解，最后取出最优的解。

2.2.2. 局限性

我们知道对于一个全排列$P(N)$，$|P(N)| = N!$。当$\mathrm{N} = 10$的时候，$N! \approx 3e7$，假设计算机每秒运算$1e6$次，求解这个问题就要运算 30 秒。而老师的模拟器中最大的$\mathrm{N} = 24$，此时$N! \approx 6e23$，求借这个问题需要$6e17$秒，而 1 年是$3e7$秒，我们穷尽一生也不可能求出他的全排列。

既然不能得到真正的全局最优解，那我们只能转而去逼近全局最优解。用比较少的搜索次数，得到一个尽可能好的结果。

3. 启发式算法——模拟退火

3.1. 引子——优化的 DFS

对于 DFS 穷举，我们可以进行贪心策略优化。先定义状态容器：

$$S = \{s_i, \dots s_{N-1}\}$$

对一个状态容器我们班有它的状态能量方程$f(S)$。

然后我们可以得到以下状态接受函数：

$$S = S_k, \quad f(S_k) < f(S_{k-1})$$
$$S = S_{k-1}, \quad otherwise$$

则状态的接受概率函数为：

$$P(k)\begin{cases}1 & ,f(S_k) < f(S_{k-1}) \\ 0 & ,otherwise\end{cases}$$

这样的剪枝方法虽然可以大大地减少搜索的次数，但是同样避免不了陷入局部最优解的情况。所以为了找到全局最优解，我们必须策略性地接受比较差的状态去寻找更优的解法。

3.2. 模拟退火

模拟退火是一种随机优化算法，其出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。

在物质温度较高时，它们的粒子热运动剧烈，粒子的位置在不停地随机变化；随着温度不断下降，物体粒子的热运动越来越弱，它们的位置也没有这么容易改变了。模拟退火就是把退火过程，放在搜索问题的求解中。

对于状态容器我们可以进行任意s_i和s_j的交换，第k次交换的结果S_k被称为S_{k-1}的相邻状态。对于退火过程，我们先定义一个初始温度T, 温度衰减系数$d(0 < d < 1)$。

则对于第k次的状态转换，我们可以定义以下状态转移函数：

$$S = S_k, \quad f(S_k) < f(S_{k-1}) \quad or \quad e^{\frac{f(S_{k-1}) - f(S_k)}{T * d^k}} > random(0,1)$$
$$S = S_{k-1}, \quad otherwise$$

从转态转移函数我们还可以得到转态转移概率函数：

$$P(k)\begin{cases}1 & ,f(S_k) < f(S_{k-1}) \\ e^{\frac{f(S_{k-1}) - f(S_k)}{T * d^k}} & ,otherwise\end{cases}$$

从状态转移函数中我们可以看出，当k越小的的时候，状态转移的发生就越容易。

在模拟退火中，我们要注意一下几点：

（1） 恰当地选择初始温度。保证开始的状态接受概率足够大。

（2） 降温系数要恰当。降温系数过快会导致模拟退火很快地收敛至一个局部最优解，从而无法找到全局最优解；降温过慢，虽然得到的解可能比较好，但是会耗费大量时间。

（3） 要确定终止条件，当到达某个足够小的温度的时候终止，或者已经陷入局部最优解时终止。

模拟退火实现的伪代码如下：

```
T, dt, eps;
while (T > eps || end_condition) {
    dE = f(S(k)) - f(S(k - 1));
    if (dE < 0)
        S(k) = S(k - 1);
    else if (exp(-dE / T) > random(0, 1))
        S(k) = S(k - 1);

    T *= dt;
}
```

4. 模拟退火在 BAP 上的应用

现在我们将模拟退火的原理具体到我们需要解决的 BAP 问题上。
下面将队模拟退火中所用到的结构和我们的 BAP 建模作一一映射。
状态容器：$V = \{v_0, \dots v_{N-1}\}$；
状态能量方程：$f(S(V))$；
初始温度：$\mathrm{T} = 2.3\mathrm{e}3$；
降温系数：$\mathrm{d} = 0.999$；
终止温度：$\mathrm{eps} = 1\mathrm{e} - 32$
算法伪代码：

$SA(T = 2.3e3, d = 0.999, eps = 1e - 32)$
$begin$
$Solution = greedy(V)$
$While(T > eps)$
$TmpV = V$
$Randomly\ swap\ 2\ element\ In\ V$
$TmpS = greedy(TmpV)$
$dE = f(TmpS) - f(Solution)$
$if(dE < 0\ or\ exp(-dE / T) > rand(0,1))$
$Solution = TmpS;$
$V = TmpV$
$T *= d;$
$Return\ Solution$
end

5. 具体实现

见附件：src 文件夹

6.实验结果

实验使用了老师所给的模拟器上的 12 个样本关卡，另外还有我自己生成的 3 个随机样例。实验结果用模拟器的贪心算法和我实现的模拟退火算法作对比。具体的关卡数据放在附件：stage 文件夹，模拟退火得到的最优解数据放在附件：solution 文件夹。最优解数据结构为：$S(V) = (B,T,U,W,D)$,这里只作 U,W,D 的对比。B 和 T 的对比可以利用模拟器得到的结果对比，自创关卡额关卡数据在附件：stage/import 文件夹。

	U		W		D		f(S)	
stage	greedy	SA	greedy	SA	greedy	SA	greedy	SA
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	20
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
My_1	16	9	32	23	15	15	1679	961
My_2	0	0	17	9	14	13	47	31
My_3	0	0	0	0	13	13	13	13

根据表格可以看出，模拟退火算法的结果比贪心算法得到的结果要好很多。在经过多次的模拟退火之后，大部分问题都得到了最优解（除了 My_1 都得到了最优解）。

7.总结

作为现代启发式算法中最基础的模拟退火，主要应用于求解现实生活中经常见到的 NP-HARD 问题的次优解，本文所涉及的 BAP 问题就是其中之一。模拟退火在 BAP 问题的求解中取得了很好的效果。实验中选择参数和多次迭代运算花费了大量的时间和精力，说明模拟退火的参数选择是非常重要的。

8.参考文献

[1]蔡文志

《束搜索应用在港口停泊位置分配问题上的研究》（2006）

[2]ThomasH. Cormen Charles E.LeisersonRonald L. Rivest Clifford Stein

《Introductionto Algorithms , Third Edition》(2013)