商人们怎样安全过河

16271012 雷佳宾

15271139 陈思言

16171061 郭宇轩

摘要：将商人过河问题转化为图论问题，再通过迪杰斯特拉算法寻找可能存在的从初始状态到最终状态的最短路径，即为安全过河的方法。

关键词：最短路径；迪杰斯特拉算法

问题描述

在河的一侧，有 N 个商人和 N 个随从需要过河，河中有一条小船，一次最多只能载重 M 人。随从们密约，在河的任一岸，一旦随从的人数多于商人的人数，就杀人越货，乘船渡河的方案由商人决定。问：商人们有安全过河的方案吗？如果有，怎样才能安全过河？

问题分析

商人们可以决定每一步（此岸到彼岸或彼岸到此岸）船上的人员，要求在安全的前提下(每一步两岸的随从人数都不比商人多),经有限步使全体人员过河。

考虑河的两岸，假设 x 为某一时刻此岸的商人数，y 为这一时刻此岸的随从数，S 为所有允许的状态的集合，那么 x 和 y 要满足的条件有：

1. $x=0$ 时，此时此岸没有商人，随从人数可以任意，故 $y\in[0,N]$；河的对岸商人数为 N，随从数不可能超过商人数，不用考虑。
2. $x=N$ 时，此岸随从数不可能超过商人数，故 $y\in[0,N]$；河的对岸没有商人，不用考虑。
3. $0<x<N$ 时，考虑此岸需要满足 $x\geq y$,考虑对岸（总人数减去此岸的人数即为对岸的人数）需要满足 $N-x\geq N-y$ 即 $x\leq y$,故此时 x 与 y 满足关系 $x=y$ 。

从而 S 可记为：

$$S=\{(x,y)|x=0,y=0,1,2\dots N;x=y=1,2\dots N-1;x=N,y=0,1,2\dots N\}$$

N 取值为 3 时用坐标表示如图 1：

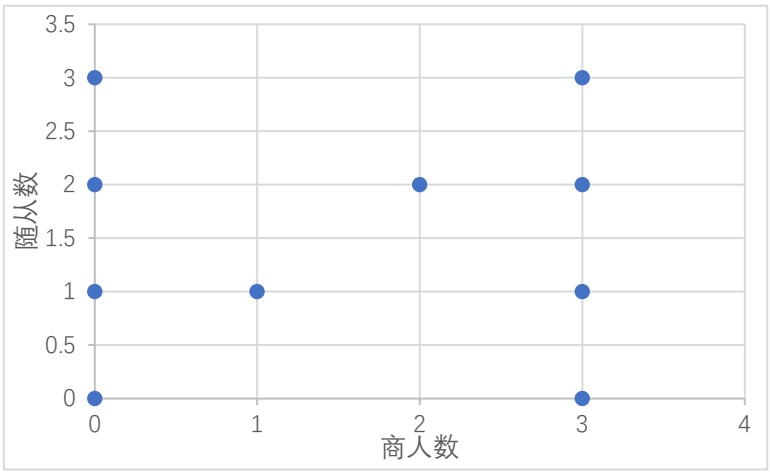

图 1

从图 1 中可以看到，初始点位于右上角，坐标为（N,N），表示此岸有 3 个商人，3 个随从；终止点位于左下角，坐标为（0，0），表示此岸没有人了。

根据 S 的表达式可知，无论 N 的取值是多少，最后的允许状态均为如图 1 所示的“N”型，这样我们在后面建图时就可以根据 N 的大小直接存这些点的坐标而不用对于每一个 N 再去具体算 S 的集合。

对于可以允许的决策，即某一次渡河时船上人数的确定，我们假设 u 为某一时刻船上的商人数，v 为这一时刻船上的随从数，D 为所有允许的决策的集合，那么 u 和 v 要满足的条件有：

1. 船上至少要有 1 个人，最多不能超过船的容量，即： $1 \le u + v \le M$
2. 船上的商人数不能小于随从数，即： $u \ge v$ ，由拓展分析 2 中所述，这一点其实在考虑所有允许的状态时已经隐含地考虑了，所以并不需要再去专门考虑。

从而 D 可记为：

$$D = \{(u,v) | u + v = 1,2 \dots M, u, v \in [0, \min(M,N)]\}$$

解题思路及建模

可以考虑将所有的状态当作顶点存起来，将一个顶点能到达的所有其他顶点用一条单向边连接起来，那么我们要找的就是从初始点（N，N）到终止点（0，0）的一条路径。

当然，遍历法是最容易想到的一种方法，即从初始点出发，遍历所有与它相连接的顶点，再从每个顶点出发继续遍历，直到找到一条到终止点的路径为止。

具体的遍历算法可以采取深度优先或广度优先来做。

采用遍历的方法还需要注意几点问题，因为船只能开往河对岸，所以它的遍历过程从坐标上来看是一次右下一次左上，即一次过河一次回来，也就是说并不是所有的有边的路径都可以走，这一点需要加以限定； 而且考虑到图中的成环问题，还需要额外去记录访问到的每个顶点的状态信息（不能简单记录每个顶点是否被访问，因为一个顶点可以被多次访问）。实际上这个代码并不好写，而且时间复杂度也很高。粗略来看，DFS 时间复杂度超过了 $O(n^2)$ ；BFS 时间复杂度为 $O(n^2)$，但是不能保证求得的解为最优解。至于穷举的算法，则达到了阶乘级的复杂度，随着数据的变大，运行的时间将急剧地增长，根本无法胜任稍微大一些的数据。

我们换个角度重新审视一下这个问题。

首先，每个顶点都有两个不同的状态，一个是从这个点出发指出该点的状态，另一个是从对岸到此岸指向该点的状态，那么我们不妨将同样的状态图 S 存成两份，分别来存储这两种状态，将其分别记为 A，B。那么一次右下一次左上的过程就变成了一次从 A 图到 B 图，一次从 B 图到 A 图的过程。

将 A、B 两图中的顶点记为不同的顶点，建边时遍历 A 图中的所有顶点，将这个顶点与其所能到达的所有顶点在 B 中对应的顶点建立单向边，这就是所谓的“一次右下”。而“一次左上”正好就是反过来的从 B 图往 A 图建立一条单向边。

比如：A 中 m 点到 B 中 n 点建立起了一条 $m->n$ 的单向边，对应“一次右下”，那么再建立一条 $n->m$ 的单向边，对应的就是“一次左上”，它们正好是对应起来的。建立的图（清晰起见，只建了前几个点对应的边）如图 2，所有的边都是双向的。

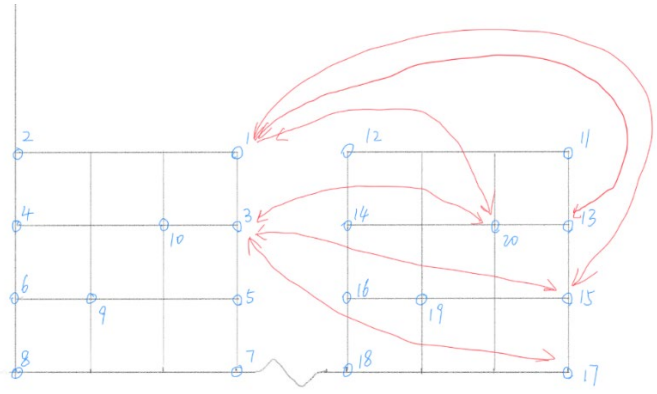

图 2

其次，联想到迪杰斯特拉算法可以求单源最短路径（并且能记录前驱结点），而且图中即使存在环也不影响算法的正确性，那么把每条边的权值设置为一个相等的常量（比如设置为 1），求出从起始点到终止点的最短路径，不断地追溯前驱结点就能找到我们要求的过河方法。

代码实现

首先，存储所有的状态，即顶点坐标，将起始点记为 1 号顶点（位于 A 图），将终止点记为 $(2 \times (N+1) + num)$ 号顶点，具体的点的标记如上面图 2 所示，其中 num 为所有允许的顶点个数：

$$num = 3 \times N + 1$$

A 图点的标记加上 num 即为 B 图中对应的点。

建边的过程与“解题思路及建模”里面所述的方法相同，然后跑迪杰斯特拉算法，从终止点不断向前回溯至初始点则可以找到具体的路径，如果回溯不到初始点，则说明安全过河的方案并不存在。

该项目的源代码以及可执行文件已上传至 GitHub，可自由查看[1]。

当人数 N 为 3，船的容量 M 为 2 时，输出结果如图 3：

```
Each of the people num is:
3
The capacity of the ship is:
2
                        The method is:
  Salesman  Follower  River(S,F)River  Salesman  Follower
     3         3      --->(0 , 2)--->      0         0
                                                     |
     3         1      <---(0 , 1)<---      0         2
     |
     3         2      --->(0 , 2)--->      0         1
                                                     |
     3         0      <---(0 , 1)<---      0         3
     |
     3         1      --->(2 , 0)--->      0         2
                                                     |
     1         1      <---(1 , 1)<---      2         2
     |
     2         2      --->(2 , 0)--->      1         1
                                                     |
     0         2      <---(0 , 1)<---      3         1
     |
     0         3      --->(0 , 2)--->      3         0
                                                     |
     0         1      <---(0 , 1)<---      3         2
     |
     0         2      --->(0 , 2)--->      3         1
                                                     |
     0         0                           3         3


11 steps in total.
请按任意键继续. . .
```

图 3

在图中我们可以很清晰直观的看到每一步的决策过程以及所用的步数，便于查看算法是否有误以及进一步的分析。

算法复杂度分析

设商人数和随从人数均为 n，船的容量为 m。

算法第一步需要记录所有可能的状态，状态总数为 n^2 但是只有$(6n+2)$个是可以去到的，此步时间和空间复杂度均为 $O(n)$。

算法第二步建边，对于稠密图来说复杂度为 $O(n^2)$，但是受船数量的限制，实际边的数量为 $O(nm^2)$ ，我们建边采用了$O(n^2)$ 的算法，但是存边用的是 $O(nm^2)$ 的邻接表，因此时间复杂度为 $O(n^2)$，而空间复杂度为 $O(nm^2)$。

算法第三步采用了 Dijkstra 算法，采用了优先队列进行优化，时间复杂度为 $O(ElgE)$，E 为所有边的数目，额外的空间消耗为 $O(n)$。 因此总的时间复杂度为 $O(n^2)$，空间复杂度为 $O(nm^2)$。

实测普通的笔记本电脑当 $n=10000$ 时仍可以较快速地运行完毕（算法本身约 1s，不包括输出，输出时间可能较长），因此本算法相对于穷举法效率有极大地提高。

数据分析

仍然设商人数和随从数均为 n，船的容量为 m。在测试代码时，我们发现，并不是所有的 n，m 组合都存在过河的策略。固定船的容量 m，作如下的分析：

m 等于 1 时，无论 n 等于多少，都没有过河的策略。这个很显然，因为船上必须保证有人，有一个人过去就必须有一个人回来，而人数最少的情况是 n 取 1 即一个商人和一个随从的情况，这样是无论如何都过不了河的；

m 等于 2 时，n 只有取 1，2，3 时是有解的，对应的最小过河步数分别为 1，5，11；

m 等于 3 时，n 取 1，2，3，4，5 时有解，对应的最小过河步数分别为 1，3，5，9，11；

注意到过河的步数全为奇数，这与过河到对岸的事实是相符的。

令人感到意外的是，当船的容量 m 大于等于 4 的时候，无论人数 n 如何变化，原问题始终是有解的，并且过河所需要的最小步数与人数呈现出近似一次函数的递增趋势，如图 4。

为了探究为什么船的容量大于等于 4 时该问题总是有解，我们将过河的过程加以输出，如图 4 是 n 为 10，m 为 4 的具体过河方法：

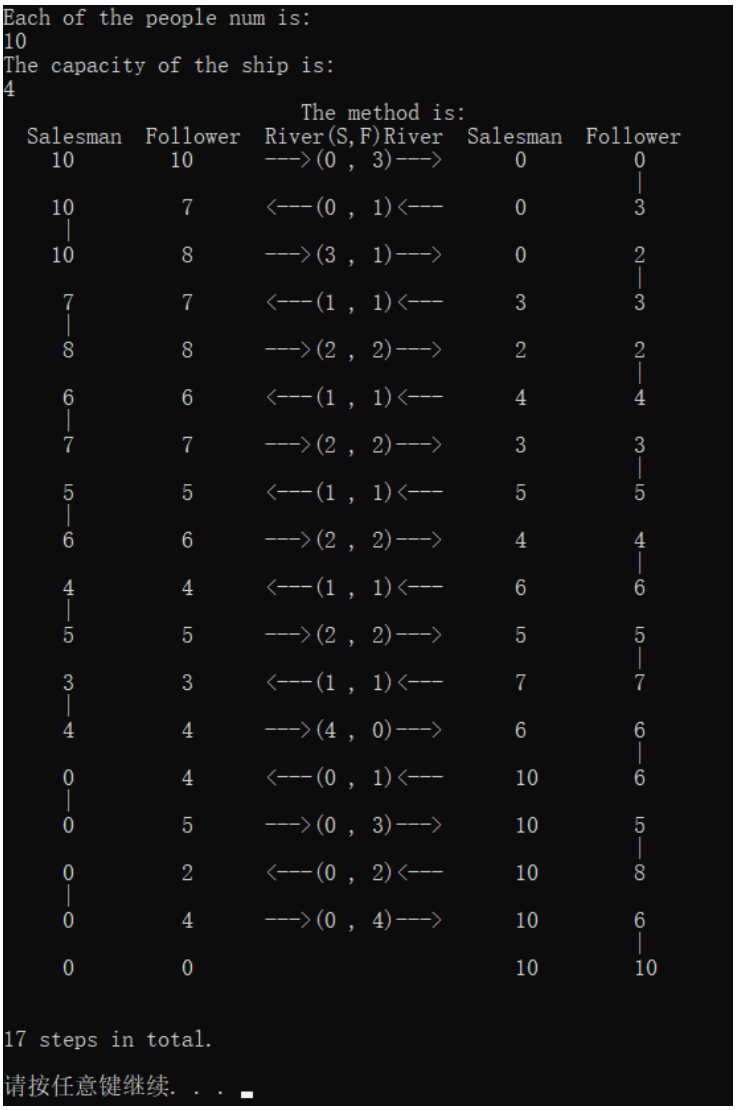

图 4

可以看到里面存在大量的 4 人去（商人和随从均为 2 人），2 人回（商人和随从均为 1 人）的情况，这样就保证了在随从数不超过商人数的情况下每次运送 1 个商人和 1 个随从到河对岸。以这种方法来过河，无论 n 有多大，都能被消解掉，最终有成功过河的方法。对于 m 大于 4 的情形，也有与之类似的方法。

图 5 展示了船的容量分别为 4，5，6，7 时，过河所需的最小步数随人数的变化趋势。

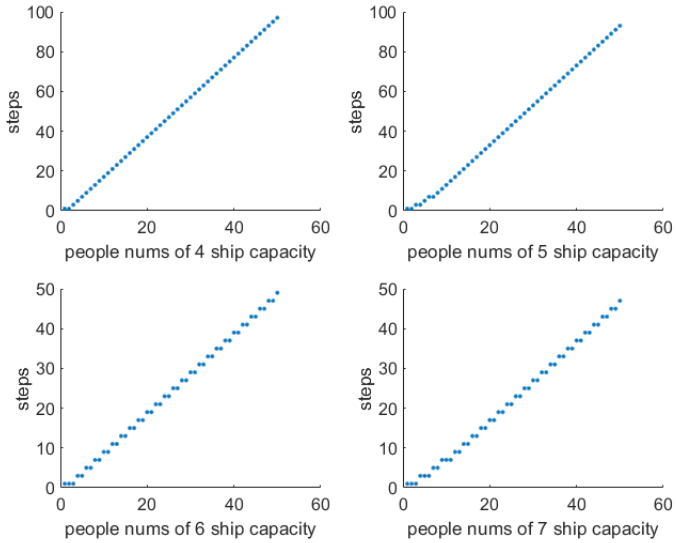

图 5

可以看出，m 为 4 和 5 的时候，当人数较大时人数与过河所需要的步骤数有较好的一次函数关系，大于 5 时则呈现出阶梯型的递增关系。并且总体的斜率是在逐渐减小的（注意纵坐标有所相同），为了验证我们的想法，进一步地，当船的容量分别为 8，9，10，11 时我们得到了图 6，可以看到阶梯型的递增关系，而且斜率确实在逐渐减小（纵坐标有所不同）：

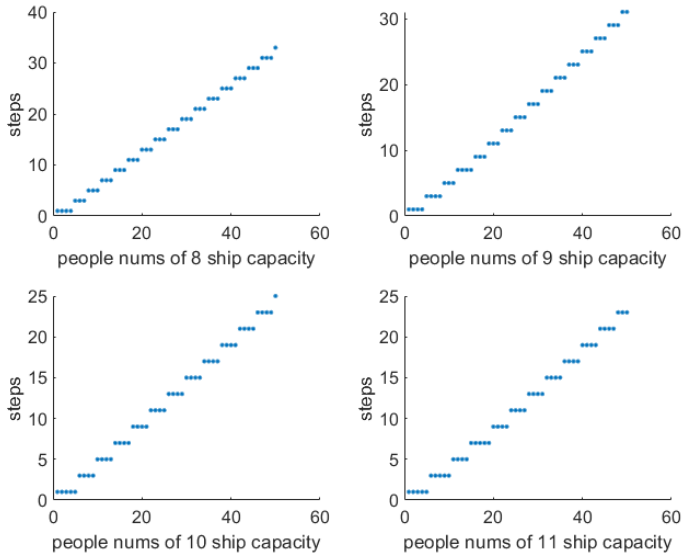

图 6

将上面所有的数据均画到一张图中，我们得到了图 7，可以很清楚的看到，m 大于 5 时的阶梯递增，并且斜率逐渐减小。

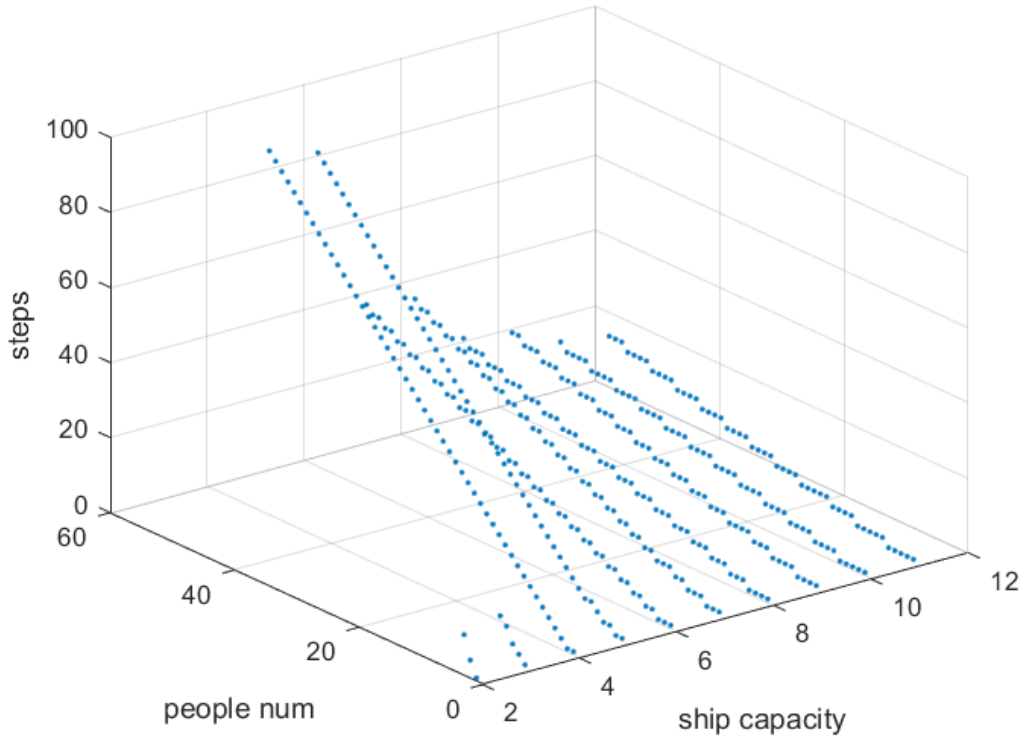

图 7

进一步扩大数据的范围，得到了图 8：

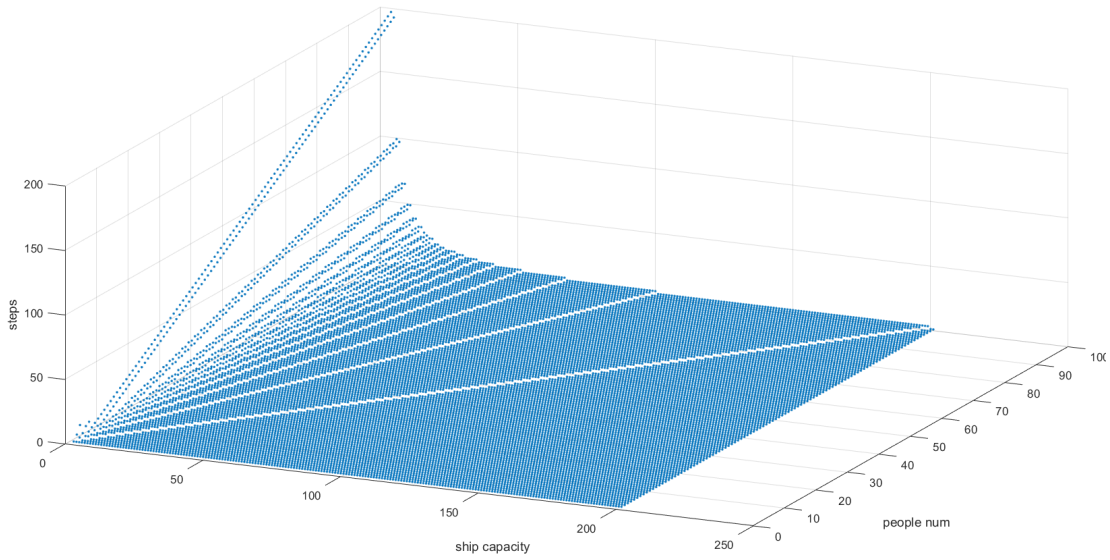

图 8

图中白色的线条从下到上分别为步数为 2，4，6，8 . . . 的情况，因为步数不能为偶数，故有这些“断层”的存在。

拓展分析

1.以上所有的讨论都是基于商人人数和随从人数相等来考虑的，那么如果商人的人数和随从的人数不相等：

当随从数多于商人数时，显然商人一上来就死了，是不可能存在解的；当商人数多于随从人数时，意味着状态图的顶点数发生变化，允许的状态更多，而建边策略没有变化，过河会更加容易，所以不再展开讨论。

2.在上面解决问题的过程中，我们只考虑了在河的两岸随从的人数不多于商人人数的情况，并未考虑船上的情况，如果考虑船上随从的人数也不能多于商人人数的情况，作如下分析：

假设出现了船上随从比商人多且商人数不为 0 的情况，设此时刻船上有 n 个商人(n 不为 0)，m 个随从，船从 A 岸驶向 B 岸，m > n，而商人和随从最开始均有 N 个人，那么为了保证到了 B 岸之后随从人数不大于商人人数，至少应该保证到了 B 岸之后人数相等,那么在这个时刻,A 岸商人的人数和随从的人数也相等。

考虑前一个时刻，即船还未出发的时刻，A 岸随从人数必然大于商人人数，商人在这个时刻就会被杀死。

所以在已经保证了两岸商人人数不小于随从人数之后，船上就不会出现商人人数小于随从人数的情况。

3.如果上述两种情况都考虑：即商人人数和随从人数不相等（商人人数大于随从人数），那么此时显然还需要考虑船上的情况，2 中的分析将不再成立。

此时图的顶点需要根据商人人数和随从人数重新建立，而建边的条件更加严格，也就意味着解的数量与人数相等时相比会更少。

总之，类似的问题均可以通过分析状态和状态转移的方式，将商人过河问题同构映射到状态图的连通性和无权图最短路径问题，这体现了本算法的通用性。当然，建顶点和建边部分的代码需要根据具体问题做出修改。

注：

[1] 项目地址位于：https://github.com/Amuvin/CrossRiver