2020 年数学建模

商人过河问题研究

姓名及学号 任昌禹 18373718

姓名及学号 胡鹏飞 18373059

姓名及学号 朱英豪 18373722

摘要

本文首先针对传统的三个商人和三个随从过河问题，建立了多步决策模型，分别采用数学图解法，深度优先搜索和广度优先搜索的方法，对模型进行了求解，给出了商人渡河的所有可行解和最优解。之后我们对模型进行了推广，研究了不同的商人数量，随从数量，以及船只容量情况下，是否有可行解以及具体的渡河方案。

针对商人过河方案的经典数学问题，本文从逻辑思考入手，考虑到问题的规模的情况，采取了编程穷举法和数学图解法，建立了基于深度优先算法的模型。考虑到该课题的逻辑性，文本同时给出了数学图解法来对初始的问题进行图形化的理解。本文采用的深度优先搜索可以搜索所有的可能解，并且在短时间内列举出来。在本文所采取的算法当中，维护了一个判断数组，用来对已经走过的状态进行记录，这很大的节省了宝贵的计算资源。同时本文在原问题的基础上进行了扩展，对若干商人和若干随从的拓展问题进行了讨论，并给出了解决扩展问题的源代码。

关键词： 状态转移、深度优先搜索、广度优先搜索

目录

1 介绍

1.1 问题重述

三个商人和三个随从准备乘船渡河，单只小船只能容纳两人，船只能由商人和随从自己划。随从们密约，在河的任意一岸，一旦随从的人数比商人多，就杀人越货。但是乘船渡河的方案由商人决定。商人们怎样才能安全渡河呢？我们将对这个问题为商人给出解决方案。

1.2 解决方案概述

将商人和随从在某一岸时的状态抽象为二维平面上的点，确定商人和随从允许的状态空间以及单步状态转移律，通过深度优先搜索方法进行枚举，以 C++编程实现。

1.3 问题扩展

若干个商人和若干个随从准备乘船渡河，单只小船能容纳若干个人，船只能由商人和随从自己划。随从们密约，在河的任意一岸，一旦随从的人数比商人多，就杀人越货。但是乘船渡河的方案由商人决定。商人们怎样才能安全渡河呢？

2 模型假设与符号定义

2.1 模型假设

为了简化我们的模型，我们提出了以下假设：

- 每个商人和随从都会划船
- 当前处在 A 岸，目标是将商人与随从均送至 B 岸

- 只有一条船，且每条船上最多只能乘坐若干个人
- 假设在船上不会发生抢劫的情况，即船上随从数可大于商人数
- 所有商人与随从之间没有矛盾，不会出现若干人不愿意坐一条船的现象
- 船从 A 岸驶至 B 岸后，下一次从 B 岸驶回 A 岸，一来一回，最终船靠在 B 岸
- 船在渡河的过程中不受外界环境的影响

2.2 符号定义

符号	*描述*
p	商人的总数
q	随从的总数
x	过河前某一岸上商人的数量
y	过河前某一岸上随从的数量
u	渡河船上商人的数量
v	渡河船上随从的数量
c	船可载人的数量
S	状态空间向量集合
D	转移律向量集合
s	状态空间向量

d	转移律向量

3 模型

3.1 原始模型

3.1.1 模型分析与建立

该问题可视作多步决策问题。每一步，船由此岸划到彼岸或者由彼岸划回此岸，都要对船上的人员进行决策，即如何安排当次渡河船上的商人和随从数，从而满足题目条件，使得两岸的随从都不比商人多。最终求解得在有限次的决策中使得所有人都到对岸去的方案。

我们首先对状态与每次决策状态进行抽象。

记第 k 次过河前此岸的商人数为x_k，随从数为y_k，其中$k=1,2,3\ldots,0\le x_k$，$y_k\le 3$。

定义状态： 将二维向量$s_k=(x_k,y_k)$定义为状态。将安全渡河状态下的状态集合定义为允许状态集合，记为$S=\{(x,y)|(x,y)$满足安全渡河的条件$\}$。

当 x 与 y 均非 0 时，当处在某一岸处，若(x,y)满足安全渡河条件，则有$x\ge y$。此时，对于另一岸处，商人、随从数分别为$3-x$、$3-y$，则有$3-x\ge 3-y$；二者联立，解得$x=y$。

若$x=0$或$y=0$，则商人与随从分离，没有杀人越货的风险，也满足安全渡河条件。

故可得：

$$S=\{(x,y)|x=0,y=0,1,2,3;\ x=y=1;\ x=y=2;\ x=3,y=0;\ x=0,y=3\}$$

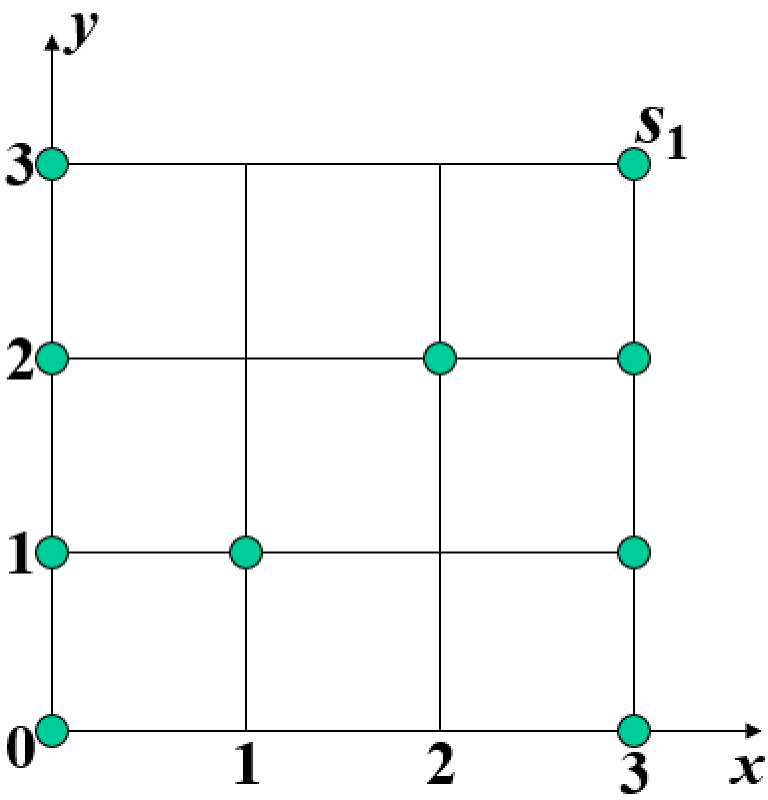

图 1 允许状态空间

记第k次渡河船上的商人数为u_k，随从数为v_k。

定义决策： 将二维向量$d_k=(u_k,v_k)$定义为决策。允许决策集合记作D。

由于每次船上载人数载人数大于 0 且小于等于 2，且u、v为大于等于0的整数，故可得：

$$\mathrm{D}=\{(\mathrm{u},\mathrm{v})|0<\mathrm{u}+\mathrm{v}\le 2,\mathrm{u},\mathrm{v}=0,1,2\}$$

状态转移律：$s_k+1=s_k+(-1)^k\times d_k$

3.1.2 模型求解

3.1.2.1 数学图解法

图解法适用于比较小型的问题。

允许决策d_k表示的是在方格中的移动，根据允许决策d_k的定义，它每次的移动范围为1~2格，并且k为奇数时向左或下方或左下方移动，k位偶数时向右或上方或右上方移动。

于是，这个问题就变成了，根据允许决策d_k，在方格中在状态（方格点）之间移动，找到一条路径，使得能从起始状态$s_1(3,3)$，到达终止状态$s_{n+1}(0,0)$。在下图中我们可以给出一种可行解，但是我们无法确定这种方法是否是最优的，这需要进一步的讨论。

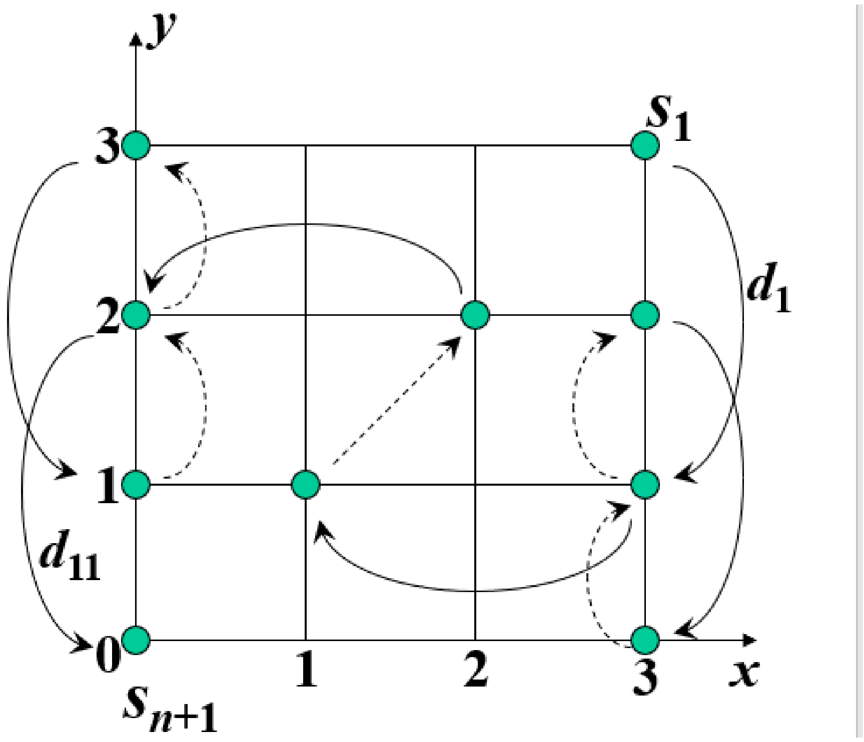

图2 其中一组可行解

可以看到我们已经给出了一种可行解。但是这种解法局限于人脑的模拟，无法

求解大型的问题，也不能利用计算来处理问题，于是我们采用编程的方法来求解这个问题。

3.1.2.2 深度优先搜索（DFS）算法

在我们遇到的一些问题当中，有些问题我们不能够确切的找出数学模型，即找不出一种直接求解的方法，解决这一类问题，我们一般采用搜索的方法解决。搜索就是用问题的所有可能去试探，按照一定的顺序、规则，不断去试探，直到找到问题的解，试完了也没有找到解，那就是无解，试探时一定要试探完所有的情况（实际上就是穷举）；

对于问题的第一个状态，叫初始状态，要求的状态叫目标状态。

我们将每一时刻的状态都视为一个节点，通过计算机的深度优先算法穷举每一种情况，如果得到了目标状态即全体成员都在对岸的情况，即结束算法。如果不能再进行状态转移则进行回溯操作，在上一个节点枚举另一种状态转移，以此类推，便可以枚举出所有情况。

3.1.2.3 广度优先搜索（BFS）算法

在本文所探讨的问题中，可以将每一个状态视为一个节点，初始状态视为一个根节点，最终状态视为目标节点，在状态和从该状态进行状态转移得到的合法的状态之间建立无向边，构成一个无向图 G，同时维护一个表用来保存已经搜索过的点，避免计算资源的浪费。

已知图$G=(V,E)$和一个源顶点 s，宽度优先搜索以一种系统的方式探寻 G 的边，从而找到 s 所能到达的所有顶点，并计算 s 到所有这些顶点的距离（最少边数），该算法同时能生成一棵根为 s 且包括所有可达顶点的宽度优先树。对从 s 可达的任意顶点 v，宽度优先树中从 s 到 v 的路径对应于图 G 中从 s 到 v 的最短路径，即包含最小边数的路径。该算法对有向图和无向图同样适用。

3.2 一般化模型的建立与求解

我们将原始模型一般化，考察 p 名商人、q 名随从、船至多载 c 人的情况。

一般化模型仅需修改允许状态集合与决策集合，其余的求解与 3.1 的原始模型的处理类似。

记第 k 次过河前此岸的商人数为x_k，随从数为y_k，其中$k = 1,2,3 \dots, 0 \leq x_k \leq x$，$0 \leq y_k \leq \mathrm{y}$。

由于在岸上的任何时刻，商人数不能少于随从数，故$p \geq q$。

当 x 与 y 均非 0 时，当处在某一岸处，若(x, y)满足安全渡河条件，则有$x \geq y$。此时，对于另一岸处，商人、随从数分别为$p - x$、$q - y$，则有$p - x \geq q - y$；二者联立，解得$0 \leq x - y \leq p - q$。

若$x = 0$或$y = 0$，则商人与随从分离，没有杀人越货的风险，也满足安全渡河条件。

故可得：

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

记第k次渡河船上的商人数为u_k，随从数为v_k。

由于每次船上载人数载人数大于 0 且小于等于c，且u、v为大于等于0的整数，故可得：

$$\mathrm{D} = \{(\mathrm{u}, \mathrm{v}) | 0 < \mathrm{u} + \mathrm{v} \leq \mathrm{c}, \mathrm{u}, \mathrm{v} \geq 0\}$$

状态转移律：$s_k + 1 = s_k + (-1)^k \times d_k$

4 模型评价

4.1 优点

我们使用不同方法对模型进行了求解。其中图解法直观，易于操作；广度优先搜索算法快速找到一条可行方案；广度优先搜索算法较快找到一条步骤数最少的最优方案。通过编程实现算法，对各种情况有一般性的解法，模型有扩展性和复现性。

4.2 不足

对于深度优先搜索算法求解时，仅对边进行染色，以确保路径的不重复。实际上，这种做法仍然会出现多次返回时（或去时）到达同一状态的情况，路径有冗余。

对于广度优先搜索算法求解时，要存储的状态较多，使得空间消耗较大。

4.3 未来工作

探讨有多条船的情况，即可连续若干次从 A 岸到 B 岸，而非当前模型条件下的船在 A、B 岸间来回。

对于深度优先搜索算法求解，增加对来时和去时的点分别进行染色，更进一步剪枝，避免路径冗余的情况。

对路径结果进行算法可视化操作，以直观地展示出每一步的决策结果。

探索商人和随从的武力差异问题，即商人可能携带一些武器，此时“状态空间”的概念发生了一些变化，并且数学图解法也不能求解该类问题，而修改穷举法的合法状态判定条件即可完成这一需求。如有些随从身材较壮而有些则不那么突出。

5 总结

我们以多种算法求解了商人过河问题。并对原始的三名商人、三名随从、限载两人的原始问题进行了扩展。我们的模型可求解 p 名商人、q 名随从、限载 c 人的一般性问题。对于各种算法的优缺点进行了分析，提出了可行的改进想法，并对后续工作进行了展望。

参考文献

1. 姜启源, 谢金星, 叶俊 (2011) 数学模型. 第四版, 高等教育出版社, 北京.

附录

1 DFS 求解（C++）

```
#include <cstdio>
#include <iostream>
using namespace std;
#define maxn 101
int NumOfMerchant, NumOfServent, CapacityOfBoat, NumOfTrans, NumOfStep;
int graph[maxn * maxn][maxn * maxn], state[maxn][maxn];
int Change_Merchant[maxn * maxn], Change_Servent[maxn * maxn];
int StepsOfMerchant[maxn * maxn], StepsOfServent[maxn * maxn];
bool flag = false;//表示是否有可行解
void print() {
    for (int i = 0; i <= NumOfStep; i++) {
        printf("(%d,%d)", StepsOfMerchant[i], StepsOfServent[i]);
        if (i != NumOfStep)
            printf(" -> ");
    }
    printf("\n");
    flag = true;
}
void DFS(int MerchantNum, int ServentNum, int step, int dir) {
    StepsOfMerchant[step] = MerchantNum, StepsOfServent[step] = ServentNum;
    if (MerchantNum == 0 && ServentNum == 0) {//如果达到的目标状态，则输出转移的
过程
        NumOfStep = step;
        print();
        return;
    }
    int FatherNode = MerchantNum * (NumOfMerchant + 1) + ServentNum;
    for (int i = 0; i < NumOfTrans; i++) {
        int NextMerchantNum = MerchantNum + dir * Change_Merchant[i];
        int NextServentNum = ServentNum + dir * Change_Servent[i];
        if (NextMerchantNum >= 0 && NextMerchantNum <= NumOfMerchant
                && NextServentNum >= 0 && NextServentNum <= NumOfServent && st
ate[NextMerchantNum][NextServentNum]) {
            int SonNode = NextMerchantNum * ( NumOfMerchant + 1 ) + NextServen
tNum;
```

```
            if (!graph[FatherNode][SonNode] && !graph[SonNode][FatherNode]) {
                graph[FatherNode][SonNode] = 1;
                graph[SonNode][FatherNode] = 1;
                DFS(NextMerchantNum, NextServentNum, step + 1, -dir);
                graph[FatherNode][SonNode] = 0;
                graph[SonNode][FatherNode] = 0;
            }
        }
    }
}
int main() {
    printf("Input: Number of the Merchant, Servant and Capacity of boat: ");
    scanf("%d %d %d", &NumOfMerchant, &NumOfServent, &CapacityOfBoat);//输入初始数据
    if (NumOfMerchant < NumOfServent) {//题目设定为商人数大于等于随从数
        printf("They can't cross the river.\n");
        return 0;
    }
    NumOfTrans = (CapacityOfBoat + 1) * (CapacityOfBoat + 2) / 2 - 1;//转移律向量的个数
    for (int i = 0; i < NumOfTrans; i++) {
        for (int j = CapacityOfBoat; j >= 1; j--) {
            for (int k = j; k >= 0; k--, i++) {
                Change_Merchant[i] = k;
                Change_Servent[i] = j - k;
            }
        }
    }
    int abs = NumOfMerchant - NumOfServent;//表示商人比随从多出来的数量
    for (int i = 0; i <= NumOfMerchant; i++) {//构造状态空间
        state[i][0] = 1;
        state[i][NumOfMerchant] = 1;     //state 数组记录可行的状态空间
        for (int j = i; j <= i + abs; j++)
            state[j][i] = 1;
    }
    DFS(NumOfMerchant, NumOfServent, 0, -1);
    //参数 1 表示商人的数量，参数 2 表示随从的数量，参数 3 表示进行的步数，参数 4 表示船行进的方向
    if (!flag)//如果没有找到可行解
        printf("They can't cross the river.\n");
```

```
}
```

2 BFS 求解（C++）

```
#include <cstdio>
#include <queue>
#define maxn 101
using namespace std;
int m, s, c;
int num;
int nn;
struct node {
    int x, y, step;
};
node foot[maxn][maxn];
node path[maxn];
int state[maxn][maxn];
bool vis[maxn][maxn];
int c_bus[maxn * maxn];
int c_fol[maxn * maxn];
int len;
void print() {
    int a = 0, b = 0;
    for (int i = len - 1; i >= 0; i--) {
        path[i].x = foot[a][b].x;
        path[i].y = foot[a][b].y;
        a = path[i].x;
        b = path[i].y;
    }
    for (int i = 0; i <= len; i++) {
        printf("(%d,%d)", path[i].x, path[i].y);
        if (i != len )
            printf(" -> ");
    }
    printf("\n");
}
int BFS() {
    queue<node> q;
```

```
    q.push((node) {
        m, s, 0
    });
    while (!q.empty()) {
        node p = q.front();
        q.pop();
        if (p.x == 0 && p.y == 0)return p.step;
        if (vis[p.x][p.y])continue;
        for (int i = 0; i < nn; i++) {
            node n;
            if (p.step % 2 != 0) {
                n.x = p.x + c_bus[i];
                n.y = p.y + c_fol[i];
            } else {
                n.x = p.x - c_bus[i];
                n.y = p.y - c_fol[i];
            }
            n.step = p.step + 1;
            if ((n.x >= 0) && (n.x <= m) && (n.y >= 0) && (n.y <= s)) {
                if (!vis[n.x][n.y] && state[n.x][n.y] == 1) {
                    q.push(n);
                    foot[n.x][n.y] = (node) {
                        p.x, p.y
                    };
                }
            }
        }
    }
    return 0;
}
void init() {
    num = m;
    nn = (c + 1) * (c + 2) / 2 - 1;
    //in case i,there are c_bus merchants and c_fols servants on thee boat
    for (int i = 0; i < nn; ) {
        for (int j = c; j >= 1; j--) {
            for (int k = j; k >= 0; k--, i++) {
                c_bus[i] = k;
                c_fol[i] = j - k;
            }
```

```
        }
    }
    int abs = m - s;
    for (int i = 0; i < num + 1; i++) {
        state[0][i] = 1;
        state[num][i] = 1;
        for (int j = i - abs; j <= i; j++)
            state[i][j] = 1;
    }
}

int main() {
    printf("Input: Number of the Merchant, Servant and Capacity of boat: ");
    scanf("%d %d %d", &m, &s, &c);
    if (m < s) {
        printf("They can't cross the river.\n");
        return 0;
    }
    init();
    len = BFS();
    if (!len)
        printf("They can't cross the river.\n");
    else print();
    return 0;
}
```