



2025 春 运筹学大作业

基于三道运筹问题的题解

作者：朱治同

时间：June 17, 2025

Email: zzt1217766450@163.com



不要以为抹消过去，重新来过，即可发生什么改变。——比企谷八幡

目录

第1章 三牛仔博弈问题	1
1.1 问题重述	1
1.2 模型假设	1
1.3 马尔可夫模型构建	1
1.3.1 状态集定义	1
1.3.2 转移概率的推导	2
1.3.2.1 吸收态自环	2
1.3.2.2 两人对决子链	2
1.3.2.3 三人对决子链	3
1.3.3 转移矩阵及转移图	3
1.4 数值计算	4
1.5 结果分析	4
1.6 小结	5
第2章 售书代理点选址问题	6
2.1 问题重述	6
2.2 数学模型构建	6
2.2.1 决策变量	6
2.2.1.1 选址变量	6
2.2.1.2 覆盖方向变量	7
2.2.1.3 覆盖状态变量	7
2.2.2 目标函数	7
2.2.3 约束条件	7
2.2.3.1 选址数量约束	7
2.2.3.2 覆盖逻辑约束	7
2.2.3.3 覆盖传播约束	7
2.2.3.4 变量域约束	8
2.3 模型求解	8
2.3.1 求解过程	8
2.3.2 最优解展示	8
2.4 小结	9
第3章 设计一个混沌系统	10
3.1 文献主要结论回顾	10
3.2 混沌系统设计	10
3.3 平衡点分析及其物理意义	10
3.4 局部线性化与稳定性条件	10
3.5 全局指数稳定性分析	11
3.6 结论	11
附录A 技术实现与实验数据汇编	12
A.1 牛仔博弈问题的概率计算	12

A.1.1 两种概率说明	12
A.1.2 吸收态概率计算	12
A.2 出版社售书模型求解技术细节	13
A.2.1 算法设计思路	13
A.2.2 算法实现	13
A.2.3 复杂度分析	13
A.2.4 计算结果数据	14
A.3 混沌系统数值验证	15
A.3.1 混沌系统可视化	15
附录 B 题解核心代码	16
B.1 牛仔博弈问题核心代码	16
B.2 售书代理点选址问题核心代码	17
B.3 混沌系统设计核心代码	18

第1章 三牛仔博弈问题

1.1 问题重述

本问题是一个博弈论问题，这个博弈过程是一个有三个牛仔参与的决斗的过程，他们分别是杰克、约翰和卡特，他们的枪法命中率分别为：杰克（Jack）80%、约翰（John）60%、卡特（Carter）40%，题目中阐释了三个牛仔射杀的规则如下：

- 每轮所有还活着的牛仔同时开枪.
- 每个牛仔只能选择一个目标（可以是其他两人中的一个）进行射击，且每轮只能开一枪.
- 牛仔们在选择射击目标时都是“理性的”，即他们会采取对自己生存最有利的策略.

1.2 模型假设

在概率博弈论中，三人决斗模型是典型的非零和博弈问题. 本题以杰克（命中率100%）、约翰（80%）和卡特（60%）构成的三方生死博弈为研究对象. 为建立可计算的数学模型，设定以下关键假设：

1. 同步射击假设：三名枪手同时开火、无先后顺序.
2. 独立事件假设：命中事件相互独立.
3. 理性最优假设：
 - (a). 当三人都在场：杰克（绝对威胁者）采取威胁消除策略，瞄准次级威胁约翰，约翰与卡特结盟，共同瞄准最大威胁源杰克. 三人不会同时阵亡.
 - (b). 当剩两人时：自然枪手互相瞄准.

1.3 马尔可夫模型构建

本题的博弈过程属于典型的马尔科夫过程，现构建本题博弈状态中的马尔科夫模型.

1.3.1 状态集定义

本题中的三名枪手的生存组合都看作马尔可夫链的“状态”，我们首先将本题的状态集（State Space）定义如下：

定义 1.1 (状态集)

定义集合 $S = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6\}$ 为状态集，其中：

$$\begin{aligned}S_0 &= ABC = \{\text{Jack}, \text{John}, \text{Carter}\} \\S_1 &= AB = \{\text{Jack}, \text{John}\} \\S_2 &= AC = \{\text{Jack}, \text{Carter}\} \\S_3 &= BC = \{\text{John}, \text{Carter}\} \\S_4 &= A = \{\text{Jack}\} \\S_5 &= B = \{\text{John}\} \\S_6 &= C = \{\text{Carter}\}\end{aligned}$$

在之后的讨论中，用字母代表人名：A 为杰克、B 为约翰、C 为卡特，后续不作说明.



1.3.2 转移概率的推导

1.3.2.1 吸收态自环

我们已成功地定义了状态集 S . 接下来, 为构建完整的马尔可夫过程, 我们引入状态转移机制, 从状态集出发, 逐步推导转移矩阵:

$$P = [P_{ij}]_{i,j=0}^6, \quad P_{ij} = \{S_{t+1} = S_j \mid S_t = S_i\} \quad (1.1)$$

其中 P_{ij} 就是 “在任意一轮 t , 处于状态 S_i 时, 下一轮 $(t+1)$ 转移到状态 S_j 的概率.

在我们的模型中, 状态 S_4, S_5, S_6 分别对应 “仅剩杰克”、“仅剩约翰” 以及 “仅剩卡特” 这三种结果——一旦对决演变到这三个状态中的任意一个, 就意味着比赛已经分出胜负, 不会再有新的枪击行为发生. 因此, 马尔可夫链中对这三个状态我们设置自环概率为 1, 表示进入后不会转出:

$$P_{44} = P_{55} = P_{66} = 1, \quad P_{4j} = P_{5j} = P_{6j} = 0 \ (j \neq i) \quad (1.2)$$

1.3.2.2 两人对决子链

对于所有非吸收态——即仍需继续射击的局面, 可分为 “仅剩两名牛仔互相对决” (两人子链) 与 “三人同时参与的对决” (三人子链) 两类情形, 二者具有不同的射击策略和转移概率结构. 我们先关注较为简单的二人对决过程.

当仅剩下两人互相对决时, 他们必然互瞄对方, 命中事件相互独立. 以下以状态 AB (杰克 vs 约翰) 为示例, 其它 “两人” 状态类推.

表 1.1: 射击概率与状态转移表

杰克击中?	约翰击中?	概率	下轮状态
否	否	$(1 - p_A)(1 - p_B) = 0.08$	AB
否	是	$(1 - p_A)p_B = 0.12$	A
是	否	$p_A(1 - p_B) = 0.32$	B
是	是	$p_A p_B = 0.48$	“双死”

其中 “双死” 结果下无人存活, 对最终赢家概率无影响, 可不作建模, 因此:

$$P_{AB \rightarrow AB} = 0.08,$$

$$P_{AB \rightarrow A} = 0.12,$$

$$P_{AB \rightarrow B} = 0.32.$$

同理可得:

$$P_{AC \rightarrow AC} = (1 - 0.8)(1 - 0.4) = 0.12,$$

$$P_{AC \rightarrow A} = (1 - 0.8) \times 0.4 = 0.08,$$

$$P_{AC \rightarrow C} = 0.8 \times (1 - 0.4) = 0.32.$$

$$P_{BC \rightarrow BC} = (1 - 0.6)(1 - 0.4) = 0.24,$$

$$P_{BC \rightarrow B} = (1 - 0.6) \times 0.4 = 0.16,$$

$$P_{BC \rightarrow C} = 0.6 \times (1 - 0.4) = 0.36.$$

1.3.2.3 三人对决子链

当三人都在场（状态 ABC）时，前文已经给出了瞄准的方案：杰克（A）瞄准约翰（B），命中率 $P_A = 0.8$ ；约翰（B）与卡特（C）均瞄准杰克（A），命中率分别 $P_B = 0.6$ 、 $P_C = 0.4$ 。

为了方便说明三人在场的博弈过程，我们引入符号 $H_{X \rightarrow Y}$ 表示枪手 X 击中 Y 的事件，存活条件通过逻辑运算符 (\cap , \cup) 和补集 (\bar{H}) 组合，得到下表：

表 1.2: 状态概率计算表

下轮状态	存活条件	概率计算	数值
ABC	$\bar{H}_{B \rightarrow A} \cap \bar{H}_{C \rightarrow A} \cap \bar{H}_{A \rightarrow B}$	$(1 - \rho_B)(1 - \rho_C)(1 - \rho_A)$	0.048
AC	$\bar{H}_{B \rightarrow A} \cap \bar{H}_{C \rightarrow A} \cap H_{A \rightarrow B}$	$(1 - \rho_B)(1 - \rho_C)\rho_A$	0.192
BC	$(H_{B \rightarrow A} \cup H_{C \rightarrow A}) \cap \bar{H}_{A \rightarrow B}$	$[1 - (1 - \rho_B)](1 - \rho_A)$	0.152
C	$(H_{B \rightarrow A} \cup H_{C \rightarrow A}) \cap H_{A \rightarrow B}$	$1 - (0.048 + 0.192 + 0.152)$	0.608

1.3.3 转移矩阵及转移图

完整转移矩阵如下：

$$P = \begin{pmatrix} 0.048 & 0 & 0.192 & 0.152 & 0 & 0 & 0.608 \\ 0 & 0.08 & 0 & 0 & 0.12 & 0.32 & 0 \\ 0 & 0 & 0.12 & 0 & 0.08 & 0 & 0.32 \\ 0 & 0 & 0 & 0.24 & 0 & 0.16 & 0.36 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (1.3)$$

将所有非零转移概率填入矩阵 P 后，为了可视化转移过程，我们可将其直接转化为马尔科夫状态转移图：

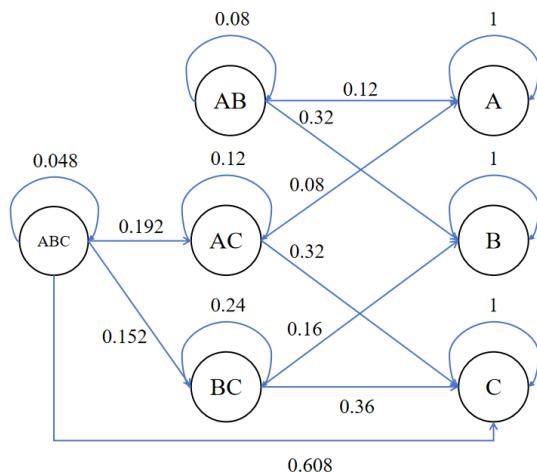


图 1.1: 三牛仔博弈状态转移图

1.4 数值计算

在1.3.3中，我们已构造好状态转移矩阵，首先，我们将所有可能的对决局面抽象为一个包含七个状态的马尔可夫链，状态顺序固定为 ABC, AB, AC, BC, A, B, C ，比赛开始时，三人都在场，对应概率向量为：

$$\nu = [1, 0, 0, 0, 0, 0, 0] \quad (1.4)$$

当轮数 $t = 1$ 时，可以得到：

$$v_1 = v_0 P = [1, 0, 0, 0, 0, 0, 0] \cdot P = [P_{ABC \rightarrow *}] = [0.048, 0, 0.192, 0.152, 0, 0, 0.608] \quad (1.5)$$

其中后三项即为第一轮结束后吸收态 A, B, C 的概率：

$$P(A) = 0, \quad P(B) = 0, \quad P(C) = 0.608. \quad (1.6)$$

一般地，第 N 轮后的分布为

$$\nu_N = \nu_0 P^N \quad (1.7)$$

我们只需关注向量 ν_N 的第 5、6、7 个分量，即

$$P(A) = \nu_{N5}, \quad P(B) = \nu_{N6}, \quad P(C) = \nu_{N7} \quad (1.8)$$

需要注意， $\nu_{N5}, \nu_{N6}, \nu_{N7}$ 并非角色在第 N 轮尚存的概率，而是系统在第 N 轮已进入该角色获胜吸收态的概率，反映“到目前为止比赛已经结束，且赢家是谁”的累积概率。若我们关注某角色是否“在第 N 轮尚未被淘汰”，需考虑所有包含该角色的非吸收状态与其吸收态的概率之和。具体解释详见A.1.1.

1.5 结果分析

为了直观了解三人存活率随轮数变化的趋势，我们计算了 $N=1, 2, 3$ 时的吸收概率，现呈现结果如下，计算完整结果详见附录A.1.2.

表 1.3: 每一轮三个牛仔的存活率

轮数 N	P(A) (杰克)	P(B) (约翰)	P(C) (卡特)
0	100.000000%	100.000000%	100.000000%
1	24.000000%	20.000000%	100.000000%
2	12.672000%	10.080000%	75.520000%
3	11.206656%	8.144640%	70.312960%

至此，我们得到了以下结论：

- 第 1 轮结束，卡特存活概率最大，因为第一轮没有牛仔射杀之，其存活概率为 100%.
- 第 2 轮结束，卡特的胜算下降至约 75%，而杰克和约翰的胜率依然偏低.
- 第 3 轮结束，卡特存活概率接近 70%，前三轮结果已经表明：“最弱者优势”在多轮对决中越发显现.

经过以上分析，在本题所研究的理性设计策略情境下，**卡特活下来的概率最大**，这是因为最强的杰克和最弱的卡特会互相牵制，而卡特通过理性的策略（射击最强的杰克）可以最大化自己的生存机会。相比之下，杰克和约翰因为枪法较好，更容易成为优先射击的目标，因此生存概率较低。

1.6 小结

该问题结果表明，在多数对抗性场景中，我们直觉认为“更强者优势越大”，但模型却告诉我们：在多方同时竞争且可选择目标的情形里，最弱的参与者反而因“被忽视”获得了生存优势。这一悖论正是博弈论中一个重要效应经典体现：**最强者先遭排挤，最弱者最终笑到最后。**



图 1.2: 决斗三人对决子链示意图（注：图片生成自豆包 AI）

第2章 售书代理点选址问题

2.1 问题重述

本题描述了某市划分为 7 个区，每区都有一定数量的大学生。根据题意，这 7 个区的邻接关系可以用如下无向图表示：

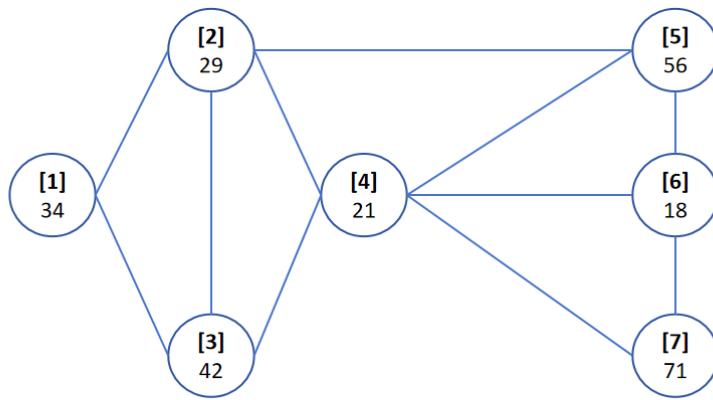


图 2.1：代售点结构图（如 [1]-34 代表一区代售点有 34 千个学生）

现拟在 7 个区域中选建 2 个代理点，每个点可服务所在区域及恰好一个相邻区，目标是最大化覆盖的大学生人数。

2.2 数学模型构建

本题属于最大覆盖模型 (**Maximum Covering Location Problem, MCLP**)，是一种用于解决设施选址问题的数学模型，旨在确定设施的最佳位置以最大化服务覆盖范围，可以用整数线性规划 (ILP) 来建模求解。构建这一数学模型，我们首先要确定决策变量，然后设置目标函数，接着添加约束条件：

2.2.1 决策变量

为了准确描述选址和覆盖关系，我们引入三组二元变量作为决策变量。

2.2.1.1 选址变量

选址变量表示是否在区域 i 设立销售代理点：

$$x_i = \begin{cases} 1, & \text{在区域 } i \text{ 设立代理点} \\ 0, & \text{否则} \end{cases} \quad \forall i \in \{1, 2, \dots, 7\}$$

2.2.1.2 覆盖方向变量

覆盖方向变量表示区域 i 的代理点是否覆盖相邻区域 j :

$$y_{ij} = \begin{cases} 1, & \text{区域 } i \text{ 的代理点覆盖区域 } j \\ 0, & \text{否则} \end{cases} \quad \forall i, j \in \{1, 2, \dots, 7\}$$

其中 $j \in N(i)$ 表示区域 j 与区域 i 相邻.

2.2.1.3 覆盖状态变量

表示区域 k 是否被至少一个代理点覆盖:

$$z_k = \begin{cases} 1, & \text{区域 } k \text{ 被至少一个代理点覆盖} \\ 0, & \text{否则} \end{cases} \quad \forall k \in \{1, 2, \dots, 7\}$$

2.2.2 目标函数

我们的目标是让尽可能多的大学生能够购买到书，因此优化目标就是最大化被覆盖区域的学生总数. 这可以表示为:

$$\max \sum_{k=1}^7 p_k z_k \quad \text{其中 } p_k \text{ 为图2.1中各区域人口基数} \quad (2.1)$$

其中， p_k 是区域 k 的学生人数（单位：千人）. 这个目标函数确保我们优先覆盖学生人数较多的区域，从而提升整体服务效益.

2.2.3 约束条件

为满足题目要求的选址约束与覆盖逻辑，我们需要建立以下四类约束.

2.2.3.1 选址数量约束

严格限定代理点数量为 2:

$$\sum_{i=1}^7 x_i = 2 \quad (2.2)$$

2.2.3.2 覆盖逻辑约束

每个代理点必须且仅能选择一个相邻区域进行覆盖（如图2.1邻接关系）：

$$\sum_{j \in N(i)} y_{ij} = x_i, \quad \forall i = 1, \dots, 7 \quad (2.3)$$

该式确保当 $x_i = 1$ 时，代理点 i 必须选择且只能选择一个邻域 j 进行覆盖.

2.2.3.3 覆盖传播约束

通过双层约束确保覆盖状态的正确传递:

$$z_i \geq \sum_{j \in N(i)} y_{ji}, \quad \forall i \quad (\text{被覆盖区域激活}) \quad (2.4)$$

$$z_j \geq y_{ij}, \quad \forall i, \forall j \in N(i) \quad (\text{直接覆盖激活}) \quad (2.5)$$

式 2.4 处理区域 i 被相邻代理点覆盖的情形，式 2.5 处理区域 j 被直接选择的情形.

2.2.3.4 变量域约束

所有决策都是二元的，仅有 0 和 1.

$$x_i, y_{ij}, z_k \in \{0, 1\} \quad \forall i, j, k \quad (2.6)$$

综上所述，我们的目的是求解下列整数规划问题：

命题 2.1 (整数规划模型)

$$\begin{aligned} & \max \sum_{k=1}^7 p_k z_k \\ & \left\{ \begin{array}{l} \sum_{i=1}^7 x_i = 2 \\ \sum_{j \in N(i)} y_{ij} = x_i, \quad \forall i = 1, \dots, 7 \\ z_i \geq \sum_{j \in N(i)} y_{ji}, \quad \forall i \\ z_j \geq y_{ij}, \quad \forall i, \forall j \in N(i) \\ x_i, y_{ij}, z_k \in \{0, 1\}, \quad \forall i, j, k \\ z_i \geq x_i, \quad \forall i \end{array} \right. \end{aligned} \quad (2.7)$$

2.3 模型求解

2.3.1 求解过程

为了求解本题，我们采用 Python 的PuLP库进行运算. 由于本题的数据规模不大，故不采用贪心算法求解，我们采用暴力枚举多次迭代求取最优解. 求解详细结果参照A.2.4. 为了得到中间过程，我们采用了图论上的 **k-组合枚举算法 (K-Nodes Combinatorial Search)**，算法逻辑详见A.2.2.

2.3.2 最优解展示

通过整数线性规划模型求解，我们获得了该区域代理网点选址的最优方案. 如表 1 所示，模型求解结果推荐在区 2 和区 7 设立代理网点，这种方案可覆盖 177 千人：

表 2.1: 最优选址方案

代理位置	覆盖方向	覆盖人数	贡献率
区 2	→ 区 5	29+56=85	48.0%
区 7	→ 区 4	71+21=92	52.0%

我们可以用一个无向图可视化这个求解结果：

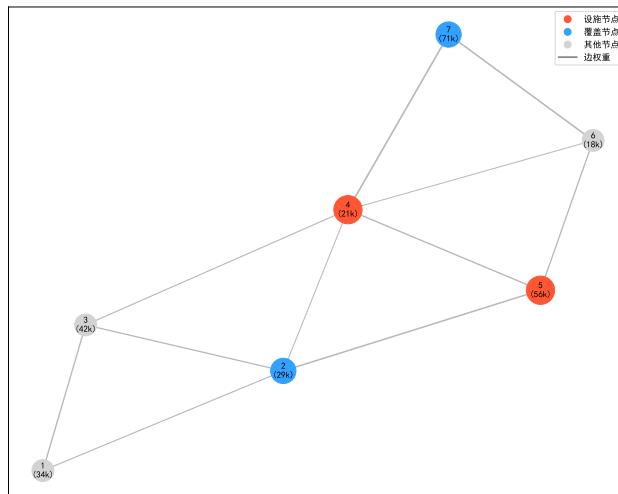


图 2.2: 最佳销售代理点安排

这张可视化图展示了一个由 7 个节点构成的网络，其中两个设施节点（红色）及其覆盖节点（蓝色）被突出显示，边宽代表连接节点的总人口数，直观呈现了最优销售代理安排方案及其覆盖的学生人数分布。

2.4 小结

为了解决本题通过构建最大覆盖模型 (MCLP) 并采用整数线性规划 (ILP) 方法，系统地解决了在 7 个区域中选址 2 个代理点以最大化覆盖大学生人数的问题。模型通过引入选址变量、覆盖方向变量和覆盖状态变量，精确描述了代理点的选址逻辑及其服务范围。目标函数聚焦于最大化覆盖的学生总数，而约束条件确保了选址数量、覆盖逻辑和变量域的合理性。通过 Python 的 PuLP 库和暴力枚举算法，我们高效地求解出最优方案：**在区 2 和区 7 设立代理点，分别覆盖区 5 和区 4，总覆盖人数达 177 千人（区 2 贡献 48.0%，区 7 贡献 52.0%）**。可视化结果进一步验证了方案的合理性，直观展示了代理点与覆盖区域的关系及人口分布。这一模型不仅提供了理论支持，也为实际决策提供了可操作的优化方案。

第3章 设计一个混沌系统

3.1 文献主要结论回顾

本题要求设计一个混沌系统并分析其平衡点的稳定性，并提供了一个文献以供参考其中的证明思路。该文作者在 Lyapunov 稳定性理论与 LaSalle 不变性原理的框架下，针对经典的三维 Lorenz 系统，提出了关于其零平衡点 S_0 全局指数稳定、全局渐近稳定与不稳定的极为简洁的代数充要条件，谨以下表说明三个定理的联系：

表 3.1: 文献对稳定性定理的分类

定理编号	参数条件	稳定性类型	判断方法	特殊点
定理 1	$c < 1$	全局指数稳定	Lyapunov 函数严格负定	收敛速度最快
定理 2	$c = 1$	全局渐近稳定（非指数）	LaSalle 不变原理	临界情况（分支点）
定理 3	$c > 1$	不稳定	一次近似线性分析	出现非零平衡点 S_+, S_-

3.2 混沌系统设计

现设计一个混沌系统：考虑经典 Rössler 系统，其以简单的结构和丰富的混沌行为而著称。在 Rössler 体系的基础上添加了 z 方向的耦合项和角振荡扰动，得到以下混沌系统：

命题 3.1 (Zhu-Chaos 系统)

在 Rössler 系统上，添加 z 耦合项和周期性扰动：

$$\begin{cases} \dot{x} = -y - z + \alpha \sin(z), \\ \dot{y} = x + ay, \\ \dot{z} = b + z(x - c) + \beta \cos(x). \end{cases} \quad (3.1)$$

此处，参数 a, b, c 控制线性项， α, β 引入周期扰动强度，为后续稳定性与分支分析提供了多维度调节手段。

3.3 平衡点分析及其物理意义

首先，我们需要明确系统的平衡态，以判断系统在扰动消失或小幅波动下的基准行为。由 $\dot{x} = \dot{y} = \dot{z} = 0$ 可得：

$$-y - z + \alpha \sin z = 0, \quad (3.2)$$

$$x + ay = 0, \quad (3.3)$$

$$b + z(x - c) + \beta \cos x = 0. \quad (3.4)$$

当 $(x, y, z) = (0, 0, 0)$ 时，第 1、2 条自动成立，第 3 条需满足 $b + \beta = 0$ 。因此，原点为平衡点的充要条件为：

$$b = -\beta. \quad (3.5)$$

若不满足此条件，需数值求解非线性代数方程才能定位其他平衡点。为突出理论可行性，后续分析基于原点。

3.4 局部线性化与稳定性条件

在确立平衡点后，我们通过线性化来刻画其邻域动力行为。对在原点处进行泰勒展开，得到雅可比矩阵：

$$J = \begin{pmatrix} 0 & -1 & -1 + \alpha \\ 1 & a & 0 \\ 0 & 0 & -c \end{pmatrix}. \quad (3.6)$$

计算其特征方程可得：

$$\det(J - \lambda I) = (-c - \lambda)(\lambda^2 - a\lambda + 1) = 0 \quad (3.7)$$

从而特征根为：

$$\lambda_1 = -c, \quad \lambda_{2,3} = \frac{a \pm \sqrt{a^2 - 4}}{2}. \quad (3.8)$$

显然，原点局部渐近稳定的充分必要条件是：

$$c > 0, \quad \Re(\lambda_{2,3}) < 0 \iff a < 0. \quad (3.9)$$

这一结果为系统参数区间的设计提供了直接指标.

3.5 全局指数稳定性分析

为了进一步确保系统在全状态空间内的收敛性，我们构建径向无界的 Lyapunov 函数：

$$V(x, y, z) = \frac{1}{2}(x^2 + y^2 + z^2) \quad (3.10)$$

对 V 取导并结合 $b = -\beta$ 条件，可得：

$$\dot{V} = -xz + \alpha x \sin z + ay^2 - cz^2 + \beta z(\cos x - 1) \quad (3.11)$$

借助不等式 $|\sin z| \leq |z|$ 与 $|\cos x - 1| \leq x^2/2$ ，可估计出：

$$\dot{V} \leq (\alpha - 1)|xz| + ay^2 - cz^2 + \frac{\beta}{2}|z|x^2 \quad (3.12)$$

为了使 \dot{V} 处处为负，参数需满足：

$$\alpha < 1, a < 0, c > \beta/2.$$

上述不等式不仅强化了局部稳定性，而且给出全局指数收敛的充分条件.

至此，我们完成了 Zhu-Chaos 系统从平衡点定位、局部与全局稳定性推导的简单流程.

3.6 结论

本文通过线性化方法和 Lyapunov 函数构造，给出了系统原点平衡点的稳定性条件：

1. 局部渐近稳定： $a < 0$ 且 $c > 0$ ；
2. 全局指数稳定：额外满足 $\alpha < 1$ 和 $c > \frac{\beta}{2}$.

参数 a, c, α, β 的分支值可通过代数关系明确表征，为混沌控制与同步提供了理论依据.

附录 A 技术实现与实验数据汇编

A.1 牛仔博弈问题的概率计算

A.1.1 两种概率说明

本题在分析三个牛仔的胜率时，涉及两种不同性质的概率度量——瞬时存活概率与吸收态累积概率。为清晰阐明二者的差异，呈下表详细说明。

表 A.1：“存活概率”与“吸收态概率”的比较

比较维度	“快照”存活概率	吸收态概率（累积胜率）
定义	在第 N 轮时刻仍存活的概率，即尚未被淘汰	截至第 N 轮，角色已获胜（进入吸收态）的累积概率
状态集合	所有非吸收状态与吸收态	仅吸收态
计算公式	Jack: $\nu_{N1} + \nu_{N3} + \nu_{N5}$ John: $\nu_{N1} + \nu_{N2} + \nu_{N6}$ Carter: $\nu_{N1} + \nu_{N2} + \nu_{N3} + \nu_{N7}$	Jack: ν_{N5} John: ν_{N6} Carter: ν_{N7}
收敛性	不一定收敛，可能波动或平稳	会收敛至最终胜率
用途	衡量比赛尚未结束时的存活概率	衡量比赛最终结果的胜出概率

两种概率度量在计算方式与收敛特性上存在本质差异，但通过马尔可夫链的稳态分析可得，Carter 的稳态胜率始终高于其他参与者。

A.1.2 吸收态概率计算

三牛仔在 N 轮累计的胜率计算结果如下表：

表 A.2: 牛仔累计胜率计算结果

N	Jack	John	Carter
1	0.000000	0.000000	0.608000
2	0.015360	0.024320	0.784064
3	0.017940	0.031324	0.816707
4	0.018286	0.033061	0.822753
5	0.018329	0.033481	0.823959
6	0.018334	0.033582	0.824217
7	0.018335	0.033606	0.824276
8	0.018335	0.033612	0.824289
9	0.018335	0.033613	0.824292
10	0.018335	0.033613	0.824293

A.2 出版社售书模型求解技术细节

A.2.1 算法设计思路

为解决该问题，我们提出了一种基于图论的 k-组合枚举搜索算法来解决该问题。该算法采用系统化的搜索策略，通过以下三个关键步骤动态构建最优设施覆盖方案：

1. **候选对生成**: 通过组合数 $C(V, 2)$ 枚举所有节点对，确保设施选址的空间完备性
2. **邻域扩展机制**: 对每个候选对 (i, k) ，生成决策空间 $\Omega = (N(i) \cup \{\emptyset\}) \times (N(k) \cup \{\emptyset\})$ ，通过选择性包含邻接节点平衡覆盖范围与计算复杂度
3. **贪心评估策略**: 在构建覆盖集 S 时实时计算目标函数值 $f = \sum w(v)$ ，通过全局比较保证解的最优性。

A.2.2 算法实现

现呈现算法伪代码：

Algorithm 1 K-组合枚举图搜索算法

Require: 节点集合 V , 邻接矩阵 A , 权重向量 w

Ensure: 最优设施配置方案 S^* 及其权重和 f^*

```

1: 初始化全局最优解  $S^* \leftarrow \emptyset, f^* \leftarrow 0$ 
2: for 所有二元组合  $(i, k) \in C(V, 2)$  do                                ▷ 遍历候选设施对
3:     生成决策空间  $\Omega \leftarrow (N(i) \cup \{\emptyset\}) \times (N(k) \cup \{\emptyset\})$ 
4:     for 每个决策  $(j, l) \in \Omega$  do                                         ▷ 穷举邻接扩展可能性
5:         构造覆盖集  $S \leftarrow \{i, k\}$ 
6:         if  $j \neq \emptyset$  then
7:              $S \leftarrow S \cup \{j\}$                                               ▷ 添加  $i$  的邻接设施
8:         end if
9:         if  $l \neq \emptyset$  then
10:             $S \leftarrow S \cup \{l\}$                                              ▷ 添加  $k$  的邻接设施
11:        end if
12:        计算目标函数值  $f \leftarrow \sum_{v \in S} w(v)$ 
13:        if  $f > f^*$  then                                                 ▷ 更新最优解
14:             $S^* \leftarrow S$ 
15:             $f^* \leftarrow f$ 
16:        end if
17:    end for
18: end for
19: return  $(S^*, f^*)$ 

```

A.2.3 复杂度分析

- 时间复杂度: $O(n^2 \cdot d^2)$ ，其中 $n = 7$, d 为平均度数 ≈ 3
- 空间复杂度: $O(n^2)$ 存储中间结果。

A.2.4 计算结果数据

本表系统展示了通过整数线性规划模型枚举求解的所有可行设施节点组合及其覆盖效果。表格数据呈现了前 30 组典型方案（完整数据含 309 种组合）。

表 A.3: Facility to Coverage Mapping (Partial Examples)

轮次	总覆盖人数	设施节点 → 覆盖节点
1	177	5 → 2
2	177	4 → 7
3	174	5 → 2
4	174	5 → 2
5	168	3 → 1
6	168	3 → 1
7	168	1 → 3
8	168	1 → 3
9	166	4 → 7
10	166	5 → 4
11	166	4 → 5
12	166	6 → 5
13	166	5 → 4
14	166	5 → 6
15	166	4 → 7
16	166	4 → 5
17	165	1 → 3
18	165	3 → 1
19	165	1 → 3
20	165	3 → 1
21	163	2 → 3
22	163	2 → 3
23	163	3 → 2
24	163	3 → 2
25	161	1 → 3
26	161	3 → 1
27	160	2 → 3
28	160	3 → 2
29	160	3 → 2
30	160	2 → 3

1. 轮次: 表示求解器尝试不同组合的迭代顺序, 编号 1 至 309 对应穷举搜索的先后次序。
2. 覆盖总人数: 反映方案的服务能力, 单位为千人 (例如 177 代表 177,000 名学生)。
3. 覆盖节点: 每个设施节点实际覆盖的区域编号 (“无”表示该节点仅覆盖自身)。
4. 设施节点: 销售代理点的具体选址位置。

A.3 混沌系统数值验证

A.3.1 混沌系统可视化

为验证本文基于 Lyapunov 方法所推导的 Scholar-Chaos 系统零平衡点局部渐近稳定性结论，我们对系统在多个不同初始条件下进行了数值仿真，并绘制了三维相图、二维平面投影图以及时间响应图。通过 Python 数值积分对系统状态轨迹进行模拟，并以 Matplotlib 进行可视化，图像直观展示了系统轨道在相空间中螺旋收敛、绕动或稳定的演化趋势，进一步支持了所提出的稳定性分析结果。

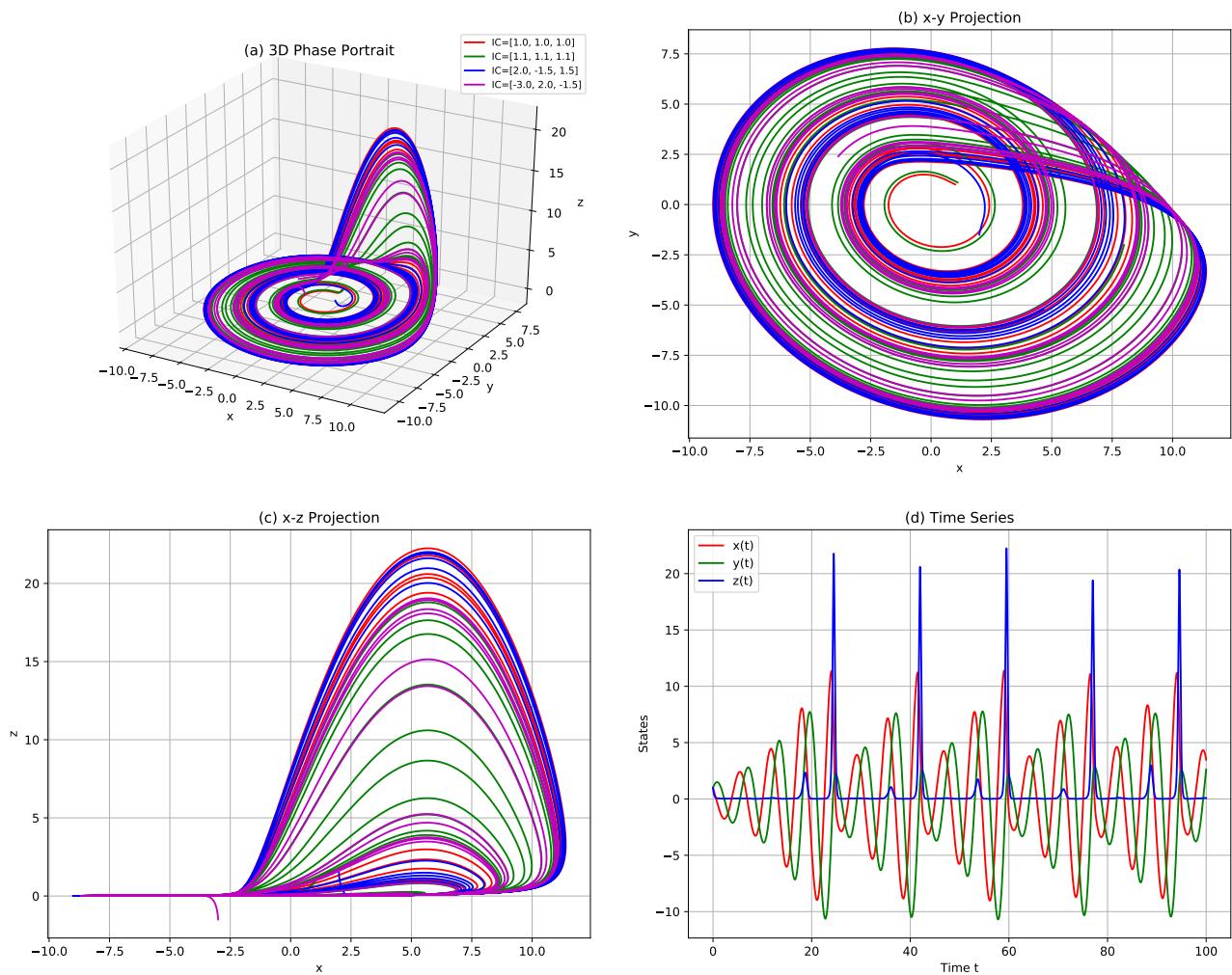


图 A.1: 混沌系统相图可视化: (a)3d 相图; (b)x-y 投影图; (c)x-z 投影图; (d) 时间响应图

附录 B 题解核心代码

- 本次运筹学大作业的完整代码已实现在 github 平台开源.
- 完整代码还包括以下内容:
 - 本题解的 L^AT_EX 项目文件.
 - 附录B中所有核心代码及其可视化实现代码.
 - 牛仔决斗模拟器.
 - 最大 Lyapunov 指数 (MLE) 计算.
- 如需查看源代码, 请点击链接: [2025-Spring-Operations-Research](#)

B.1 牛仔博弈问题核心代码

```
1 import numpy as np
2
3 def build_transition_matrix():
4     P = np.zeros((7,7))
5     # 吸收态自环
6     # A (index 4), B(index 5), C(index 6)
7     P[4,4] = 1.0
8     P[5,5] = 1.0
9     P[6,6] = 1.0
10
11    # 两人对决子链
12    # AB 对决 (索引 1 -> 1,4,5)
13    pA, pB = 0.8, 0.6
14    P[1,1] = (1-pA)*(1-pB)
15    P[1,4] = (1-pA)*pB
16    P[1,5] = pA*(1-pB)
17
18    # AC 对决 (索引 2 -> 2,4,6)
19    pC = 0.4
20    P[2,2] = (1-pA)*(1-pC)
21    P[2,4] = (1-pA)*pC
22    P[2,6] = pA*(1-pC)
23
24    # BC 对决 (索引 3 -> 3,5,6)
25    P[3,3] = (1-pB)*(1-pC)
26    P[3,5] = (1-pB)*pC
27    P[3,6] = pB*(1-pC)
28
29    # 三人对决子链
30    # ABC (索引 0 -> 0,2,3,6)
31    # A瞄准B; B、C瞄准A
32    P[0,0] = (1-pB)*(1-pC)*(1-pA)      # 保留 ABC
33    P[0,2] = (1-pB)*(1-pC)*pA          # 剩 AC
34    P[0,3] = (1 - (1-pB)*(1-pC))*(1-pA) # 剩 BC
```

```

35     P[0,6] = 1 - (P[0,0] + P[0,2] + P[0,3]) # 剩 C
36
37     return P
38
39 def compute_survival_probs(N_values):
40     P = build_transition_matrix()
41     v0 = np.zeros(7)
42     v0[0] = 1.0 # 初始在状态 ABC
43
44     results = {}
45     for N in N_values:
46         # 计算 P^N
47         PN = np.linalg.matrix_power(P, N)
48         vN = v0.dot(PN)
49         # 吸收态概率
50         pA, pB, pC = vN[4], vN[5], vN[6]
51         results[N] = (pA, pB, pC)
52
53     return results
54
55 if __name__ == "__main__":
56     Ns = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100, 1000]
57     surv = compute_survival_probs(Ns)
58
59     print("轮数 N | P(A)=Jack存活 | P(B)=John存活 | P(C)=Carter存活")
60     print("-----|-----|-----|-----")
61     for N in Ns:
62         pA, pB, pC = surv[N]
63         print(f"{N:>7d} | {pA:13.4f} | {pB:13.4f} | {pC:16.4f}")

```

B.2 售书代理点选址问题核心代码

```

1 import networkx as nx
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import itertools
5 import matplotlib
6
7 # 数据建立
8 nodes = [1, 2, 3, 4, 5, 6, 7]
9 populations = {1: 34, 2: 29, 3: 42, 4: 21, 5: 56, 6: 18, 7: 71}
10 adjacency = {
11     1: [2, 3],
12     2: [1, 3, 4],
13     3: [1, 4],
14     4: [2, 3, 5, 6, 7],
15     5: [2, 4, 6],
16     6: [4, 5, 7],
17     7: [4, 6]
}

```

```

18 }
19
20 # 暴力搜索法寻找最佳位置和覆盖范围
21 best = {'pair': None, 'covers': None, 'total_pop': 0}
22
23 for i, k in itertools.combinations(nodes, 2):
24     choices_i = adjacency[i] + [None]
25     choices_k = adjacency[k] + [None]
26     for j in choices_i:
27         for l in choices_k:
28             covered = {i, k}
29             if j: covered.add(j)
30             if l: covered.add(l)
31             total = sum(populations[n] for n in covered)
32             if total > best['total_pop']:
33                 best = {'pair': (i, k), 'covers': (j, l), 'total_pop': total}
34
35 # 提取最佳结果
36 (i, k) = best['pair']
37 (j, l) = best['covers']
38 covered_nodes = {i, k} | ({j} if j else set()) | ({l} if l else set())
39
40 # 准备 DataFrame
41 df = pd.DataFrame({
42     'Facility Node': [i, k],
43     'Covered Adjacent': [j if j else 'None', l if l else 'None']
44 })
45
46 print("最佳销售代理安排")
47 print(df)
48 print(f"\n最大覆盖学生人数: {best['total_pop']}")# Visualization
49 G = nx.Graph(adjacency)
50 pos = nx.spring_layout(G, seed=42)

```

B.3 混沌系统设计核心代码

```

1 import numpy as np
2 from scipy.integrate import solve_ivp
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.mplot3d import Axes3D
5
6
7 def modified_rossler(t, state, a=0.2, b=0.5, c=5.7, alpha=0.1, beta=0.1 ):
8     x, y, z = state
9     dx = -y - z + alpha * np.sin(z)
10    dy = x + a * y
11    dz = b + z*(x - c) + beta * np.cos(x)
12    return [dx, dy, dz]

```

```

13
14 initial_conditions = [
15     [1.0, 1.0, 1.0],
16     [1.1, 1.1, 1.1],
17     [2.0, -1.5, 1.5],
18     [-3.0, 2.0, -1.5]
19 ]
20
21
22 t_span = (0, 100)
23 t_eval = np.linspace(t_span[0], t_span[1], 10000)
24 solutions = []
25 for ic in initial_conditions:
26     sol = solve_ivp(modified_rossler, t_span, ic, t_eval=t_eval,
27                      args=(0.2, 0.2, 5.7, 0.1, 0.1), rtol=1e-8, atol=1e-10)
28     solutions.append(sol)
29
30
31
32 plt.figure(figsize=(15, 12))
33 colors = ['r', 'g', 'b', 'm']
34
35
36 ax1 = plt.subplot(221, projection='3d')
37 for sol, color, ic in zip(solutions, colors, initial_conditions):
38     ax1.plot(sol.y[0], sol.y[1], sol.y[2], color=color, label=f"IC={ic}")
39 ax1.set_title("(a) 3D Phase Portrait")
40 ax1.set_xlabel("x")
41 ax1.set_ylabel("y")
42 ax1.set_zlabel("z")
43 ax1.legend(prop={'size': 8})
44
45
46 ax2 = plt.subplot(222)
47 for sol, color in zip(solutions, colors):
48     ax2.plot(sol.y[0], sol.y[1], color=color)
49 ax2.set_title("(b) x-y Projection")
50 ax2.set_xlabel("x")
51 ax2.set_ylabel("y")
52 ax2.grid(True)
53
54
55 ax3 = plt.subplot(223)
56 for sol, color in zip(solutions, colors):
57     ax3.plot(sol.y[0], sol.y[2], color=color)
58 ax3.set_title("(c) x-z Projection")
59 ax3.set_xlabel("x")
60 ax3.set_ylabel("z")
61 ax3.grid(True)

```

```
62  
63  
64 ax4 = plt.subplot(224)  
65 sol = solutions[0]  
66 ax4.plot(sol.t, sol.y[0], label="x(t)", color='r')  
67 ax4.plot(sol.t, sol.y[1], label="y(t)", color='g')  
68 ax4.plot(sol.t, sol.y[2], label="z(t)", color='b')  
69 ax4.set_title("(d) Time Series")  
70 ax4.set_xlabel("Time t")  
71 ax4.set_ylabel("States")  
72 ax4.legend()  
73 ax4.grid(True)  
74  
75  
76 plt.tight_layout(pad=3.0)  
77 plt.savefig("Zhu_Chaos_Combined.pdf", format='pdf', dpi=300, bbox_inches='tight')  
78 plt.show()
```