卒業研究報告書

題目

囚人のジレンマ問題における最適戦略の考察

指導教員

石水 隆　講師

報告者

10-1-037-0181

竹内 亮太

近畿大学理工学部情報学科

平成 28 年 1 月 29 日提出

概要

ゲームの理論における概念の一つとして、囚人のジレンマ[1]というジレンマを扱う問題がある。囚人のジレンマとは、対戦相手がいるゲームにおける意思決定のモデルの一つであり、対戦相手の戦略が何であれ、自分の利得が大きくなるはずの戦略が結果としてお互いに大きな損害を与えているというジレンマを表した現象である。

囚人のジレンマ問題の応用問題に競りゲームがある。2 人のプレイヤーが互いに数を出し合い、数の低い方に出した数と同じ得点が入るといったゲームである。

本研究では、競りゲームに対して、その最適戦略を考察する。最適戦略を導くために、戦略をいくつか用意して各戦略間で対戦させてその有用性を検証する。

目次

]

1　序論

1.1. 囚人のジレンマ

ある犯罪者を共同で犯した嫌疑を受けている二人が警察に逮捕され、独房に入れられている。各容疑者が取りえる行動は自白するか黙秘するかどちらかであり、各容疑者は各々自分の行動及びパートナーの行動により起こる結果を知っている。結果とは(1)もし容疑者の一人が自白し、パートナーが自白しないならば、自白した者は自由になり、もう一人は20よって年の刑に服する。(2)もし両方の容疑者が自白するならば、彼らは共に5年の刑に服する。(3)もし両方の容疑者が黙秘するならば、彼らは1年ばかりの刑に服する。このような条件のもとで容疑者たちはどうするのかといった問題に対して容疑者の一人の見地から眺めてみるとする。パートナーが自白すると想定すると、自分は黙秘して 20 年刑務所に入るか、自白して 5 年刑務所に入るかであるため最適な選択として自白する。また、パートナーが黙秘した場合は、自分も黙秘して 1 年の服役もしくは自白して自由になるかである。この場合も懲役年数が短い自白する方を選ぶ。つまりパートナーの行動に関係なく自白した方が有利となるため、二人共自白してしまい、5 年刑務所に入ることになる。このように、お互いが協力して黙秘をすれば 1 年で済んだものを、協力せずに自分の利益を求めることで結果として悪い結果になってしまうというジレンマが、 A.W.Tucker により定式化されたゲーム理論[2]の一つである囚人のジレンマ[3]である。本研究では、この囚人のジレンマを応用したゲームについて研究を行う。

1.2. ナッシュ均衡とパレート最適

多人数ゲームにおいて、複数のプレイヤーが、各自の利益を最大にするように行動したとき、その行動は最終的にナッシュ均衡と呼ばれる状態になる。ナッシュ均衡[3]とは相手が戦略を変えない限り、自分の戦略の変更によって利得を増すことができない状態にあることをいう。

囚人のジレンマ問題においてナッシュ均衡を説明すると、

(1)プレイヤー＝囚人１，囚人２

(2)戦略＝｛自白する，黙秘する｝

(3)利得関数は表１のように示す

表 1

		囚人2 自白	囚人2 黙秘
囚人1	自白	(5，5)	(0，20)
	黙秘	(20，0)	(1，1)

囚人 2 が自白を選ぶならば、それに対する囚人 1 の最適選択は自白を選ぶことであり、同様に囚人 2 が黙秘を選ぶ場合にも最適戦略は自白することである。囚人２の最適選択も同様に考えられる。したがって、相手の行動を所与として自らの最適戦略を選んでいる点は囚人 1，囚人 2 ともに自白するを選んだ組であり[5]、これがナッシュ均衡である。

また、パレート最適[6]とは相手の利得を犠牲にしなければ自分の利得を改善できない状態をいう。表 1 の状態で表すと、片方が自白、もう片方が黙秘、もしくは両方が黙秘した場合がパレート最適な状態といえる。

]

1.3. 囚人のジレンマ問題における戦略

前節で述べた通り、囚人のジレンマ問題では両者が自分の利益を最大とする行動を取ると両者共に最低限の利益しか得られない。

囚人のジレンマ問題に対する既知の戦略としては、相手が何を出しても自分への被害が小さい選択をする合理的な戦略。もしくは、お互いに協力し損害を最小限にする選択を狙う非合理な戦略の２つ[7]であるが、繰り返しゲームと呼ばれる同じゲームが繰り返し行われる際に結果が変化する[9]。全部自白をする戦略、全部黙秘をする戦略、自白と黙秘を混ぜた戦略に分けることができる。[10]

1.4. 競りゲーム

競りゲームは囚人のジレンマ問題のバリエーションの一つである。2 人対戦ゲームで互いに数を出し合い、小さい数を出した方が出した数の得点を得る勝敗が決まるゲームである。

競りゲームの詳細については第 2 章で述べる。

1.5. 本研究の目的

本研究では、競りゲームにおける戦略を追求する。競りゲームの条件下において、勝率が高く、より高い点数を得るにはどのような戦略を用いれば良いのかを考察することが目的である。

1.6. 本報告書の構成

本報告書では第 2 節では競りゲームの基本的なルールについて述べる。また、第 3 節では本アプリケーションを作成した際の仕様、用意した 5 つの戦略について説明する。

2 競りゲーム

2.1. ルール

競りゲームは囚人のジレンマ問題の一種である。

競りゲームのルールは以下通りである。2 人のプレイヤーが 1～10 までの数を一つ同時に出すとき、より小さい数を出した方が出した数の得点を得る。両者の出した数が同じ場合は、両者共に出した数の 1/2 の点を得る。これを何回か行い、各自が得る点をできるだけ高くすることがゲームの目的である。表 1 に、競りゲームにおいて両者が得る点を示す。競りゲームにおける注意点として、相手に勝つことがゲームの目的では無いことが挙げられる。つまり、相手よりも多く点を得たり、相手に点を与えないようにしたりすることはゲームの勝敗には関係せず、自分が得た点のみが問題となる。

両者が得る点の期待値は、両者が共に 10 を出した時に最も高くなる。しかし、得点を得るためには相手より低い数を出す必要があるため、ミニ・マックス法に従うと最終的には両者が 1 を出す結果に収束し、両者が得られる結果は最低のものになってしまう。ミニ・マックス法とは、相手が最善手を出してくるとき、こちらが最も有利になるような手を出すことである。言い換えると自分に損がないように確実に利益が狙えるようにすることであり、競りゲームにおいて、相手に負けないように確実に得点が狙える「1」を出しつづけるという考え方である。[8]

]

表 2 競りゲームの得点表

(m,n)	10	9	8	7	6	5	4	3	2	1
10	(5,5)	(0,9)	(0,8)	(0,7)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)
9	(9,0)	(4.5,4.5)	(0,8)	(0,7)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)
8	(8,0)	(8,0)	(4,4)	(0,7)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)
7	(7,0)	(7,0)	(7,0)	(3.5,3.5)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)
6	(6,0)	(6,0)	(6,0)	(6,0)	(3,3)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)
5	(5,0)	(5,0)	(5,0)	(5,0)	(5,0)	(2.5,2.5)	(0,4)	(0,3)	(0,2)	(0,1)
4	(4,0)	(4,0)	(4,0)	(4,0)	(4,0)	(4,0)	(2,2)	(0,3)	(0,2)	(0,1)
3	(3,0)	(3,0)	(3,0)	(3,0)	(3,0)	(3,0)	(3,0)	(1.5,1.5)	(0,2)	(0,1)
2	(2,0)	(2,0)	(2,0)	(2,0)	(2,0)	(2,0)	(2,0)	(2,0)	(1,1)	(0,1)
1	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(0.5,0.5)

2.2. 戦略

本研究では、競りゲームにおける最適な戦略を検証するために 5 つの戦略を用意し、互いに対戦させる。用意した 5 つの戦略とその内容を以下に示す。

- **RND 戦略**

RAND 戦略はランダム(1～10)に数を出す戦略である。RAND 戦略では、1～10 の数を各 1/10 の確率で出す完全なランダム戦略であり、他の戦略の優劣の検証用として用いる。

- **LOW 戦略**

LOW 戦略は、低めの数(1～5)を出す戦略である。1～5 をランダム(それぞれ 20%)で出す。低めの数を出すことで勝率を上げ地道に得点を重ねていく戦略である。

- **MID 戦略**

MID 戦略は真ん中の数(3～7)を出す戦略である。3～7 をランダム(それぞれ 20%)で出す。

- **HI 戦略**

HI 戦略は高めの数(5～10)を出す戦略である。5～10 をランダム(それぞれ 20%)で出す。勝率を考えず高得点を狙う戦略である。

- **-1 戦略**

-1 戦略は前回相手が出した数を-1 した数を出す戦略である。-1 戦略では、初回はランダムで数を出す。2 回目以降では、前回相手が出した数-1 を出す。相手が 1 を出していた場合は 1 を出す。

3 結果・考察

本研究では、付録 1 に示すプログラムを用いて 5 つの戦略に対し、総当たり戦で 3 回戦をワンセットとして 100 回ずつ対戦させ、それぞれの勝率と合計得点を調べた。表 1 にその結果を示す。また、各戦略間で得た相手の戦略ごとの合計得点を表 2 に示す。

]

表 3 5 つの戦略の合計得点と勝率

戦略	RND	SMALL	MIDLE	HIGH	-1
合計得点	520	593	626	573	577
勝率	0.41	0.62	0.49	0.32	0.57

表 4 各戦略間での得点

	RND	LOW	MID	HI	-1
RND	611	200	561	748	481
LOW	593	334	820	860	357
MID	816	177	677	934	525
HI	752	62	358	1024	669
-1	553	311	597	970	453

表 1 から、戦略 MID が合計得点が最も高く、また他の戦略と比較しても勝率に大きな差がないため、戦略 MID が最も効率よく得点を稼げる戦略といえる。

一方、表 2 より、各プレイヤーが得る得点は、双方が戦略 LOW を採ったときよりも、双方が戦略 HI を採ったときの方が高いことが示される。すなわち、競りゲームでは単純に双方が利益を追求するよりも、協力しあった方が得点が高くなる。しかし、相手の出方が分からないためこのような得点になるのは不可能であると考える。相手の出方が分からない以上、初めは最も効率の良い戦略 MID を用い、中盤もしくは終盤に相手の出方や性格を考察し、それにあった戦略に変更することが、より高い得点を取る方法であると考える。

本研究では、各対戦では途中で戦略を変更することなく、最後まで同一の戦略を用いている。しかし、相手の出した手から相手の戦略を推定し、それに合わせて対戦中に戦略を変更することで結果も大きく異なると考えられる。

4 結論・今後の課題

本研究では、囚人のジレンマ問題のバリエーションの一つである競りゲームにおいて最適な戦略の検証を行った。本研究では競りゲームに対して 5 つの戦略を用意し、その中での最適解を見つけることができた。しかしこの競りゲームは不完全情報ゲームであるため、不確定な情報や人間的要素が多く存在する。性格にあった戦略を考察し、より多くの戦略に対して実験を行い、最も最適な戦略を探し出すことが今後の課題となる。

]

参考文献

[1] 大澤博隆，今井倫太，人間同士による繰り返し型囚人のジレンマゲームの実施と結果，ゲームプログラミングワークショップ 2007 論文集，Vol.2007，No.12，pp188-194(2007)，
http://id.nii.ac.jp/1001/00097658/

[2] 小島寛之，松原望 著：戦略とゲームの理論，東京図書（2011）

[3] Morton D.Davis 著,桐谷維,森克美 訳：ゲームの理論入門－チェスから核戦争まで，講談社（1973）.

[4] 中山幹夫著：協力ゲームの基礎と応用，勁草書房（2012）

[5] 川又邦雄著：ゲーム理論の基礎，培風館（2012）

[6] William Poundston 著，松浦俊輔 訳：囚人のジレンマ：フォン・ノイマンとゲームの理論，青土社（1995）

[7] 福田陽介，有限繰り返し囚人のジレンマにおける協力行動，東京工業大学大学院社会工学専攻 平成 24 年度修士論文、(2013)，
http://www.soc.titech.ac.jp/info/docs/11M43371.pdf

[8] Jason Fox : Tic Tac Toe: Understanding The Minimax Algorithm, Never Stop Building LLC（2013/12/13），http://neverstopbuilding.com/minimax

[9] 吉浦賢，繰返し囚人ジレンマにおける臨界現象，高知工科大学 情報システム工学科，平成 19 年度学位学士論文，(2008)，
http://www.kochi-tech.ac.jp/library/ron/2007/2007info/1080425.pdf

[10] 行方常幸，囚人のジレンマゲームにおける協調行動とプレイヤーの合理性，数理解析研究所講究録，Vol.1194，pp.47-55，(2001)，
http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1194-7.pdf

]

謝辞

本研究を行うにあたり、直接指導していただいた近畿大学理工学部情報学科情報論理工学研究室石水講師には大変お世話になり、日頃の研究に関する議論やレポート、研究へのアドバイス、論文指導に対し適切なご助言と励ましなどを頂きましたので、ここに感謝の意を表します。

]

付録１

本研究で作成した競りゲームのソースプログラムを以下に示す。

```
package seri;

import java.util.Random;

public class game {
    int m; // 自分
    int n; // 相手

    /* 戦略の種類 */
    // ランダムに数字を出す戦略
    public double random() {
        Random rnd = new Random();
        double ran = rnd.nextInt(10) + 1;
        return ran;
    }

    // 低い数値(1～5)しか出さない戦略
    public int small() {
        Random rnd = new Random();
        int ran = rnd.nextInt(5) + 1;
        return ran;
    }

    // 真ん中の数値(3~7)しか出さない戦略
    public int midle() {
        Random rnd = new Random();
        int ran = rnd.nextInt(5) + 3;
        return ran;
    }

    // 高い数値(5~10)しか出さない戦略
    public int high() {
        Random rnd = new Random();
        int ran = rnd.nextInt(5) + 5;
        return ran;
    }
```

]

```
    /* mの得点 */
    public double hikakuM(double m, double n) {
        if (m < n) {
            return m;
        } else if (m == n) {
            return m / 2;
        } else {
            return 0;
        }
    }

    /* nの得点 */
    public double hikakuN(double m, double n) {
        if (m > n) {
            return n;
        } else if (m == n) {
            return n / 2;
        } else {
            return 0;
        }
    }

    public static void main(String[] args) {
        game game = new game();
        game.random();
        double kekkaM100 = 0; //100 回勝負した時のMの勝った回数
        double kekkaN100 = 0;
        double M_shouritu = 0; //Mの勝率
        double N_shouritu = 0;

        for(int i=1;i<=100;i++){
            System.out.print("\n" + i + "回目の勝負 : \n");


            System.out.print("\n1 回目 : ");
            double m1 = game.random();      //戦略をいれる
            double n1 = game.random();      //戦略をいれる
            System.out.printf("m=%4.0f ", m1);
            System.out.printf(", n=%4.0f ", n1);
            double kekkaM1 = game.hikakuM(m1, n1);
            double kekkaN1 = game.hikakuN(m1, n1);
            System.out.printf(" → mの得点 : %4.1f , nの得点 : %4.1f" , kekkaM1,kekkaN1);
```

]

```
            System.out.print("\n2 回目 : ");
            double m2 = game.random();      //戦略をいれる
            double n2 = game.random();      //戦略をいれる
            System.out.printf("m=%4.0f ", m2);
            System.out.printf(", n=%4.0f ", n2);
            double kekkaM2 = game.hikakuM(m2, n2);
            double kekkaN2 = game.hikakuN(m2, n2);
            System.out.printf(" → mの得点 : %4.1f , nの得点 : %4.1f" , kekkaM2,kekkaN2);

            System.out.print("\n3 回目 : ");
            double m3 = game.random();      //戦略をいれる
            double n3 = game.random();      //戦略をいれる
                System.out.printf("m=%4.0f ", m3);
            System.out.printf(", n=%4.0f ", n3);
            double kekkaM3 = game.hikakuM(m3, n3);
            double kekkaN3 = game.hikakuN(m3, n3);
            System.out.printf(" → mの得点 : %4.1f , nの得点 : %4.1f" , kekkaM3,kekkaN3);


            double kekkaM = kekkaM1 + kekkaM2 + kekkaM3;
            double kekkaN = kekkaN1 + kekkaN2 + kekkaN3;
            kekkaM100 += kekkaM;
            kekkaN100 += kekkaN;


            System.out.printf("\nmの合計得点 : %4.1f , nの合計得点 : %4.1f\n", kekkaM,kekkaN);
            if (kekkaM > kekkaN) {
                System.out.print("mの勝ち\n");
                M_shouritu++;
            } else if (kekkaM < kekkaN) {
                System.out.print("nの勝ち\n");
                N_shouritu++;
            } else {
                System.out.print("引き分け\n");
            }
            System.out.printf("\nMの合計得点：%4.1f\n",kekkaM100);
            System.out.printf("\nNの合計得点：%4.1f\n",kekkaN100);
            System.out.printf("\nMの勝率：%4.3f\n",M_shouritu/100);
            System.out.printf("Nの勝率：%4.3f\n",N_shouritu/100);
        }
    }
    }
```