Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

Лабораторная работа №3 на тему "Неантагонистические игры. Критерии выбора оптимальных стратегий в бескоалиционных играх нескольких игроков"

по дисциплине «Теория игр и исследование операций»

Вариант 12

Студент ИУ8-104 (Группа) Мильченко И. Д. (И. О. Фамилия) _____ (Подпись, дата)

Преподаватель Коннова Н. С. (И. О. Фамилия) _____ (Подпись, дата)

Москва, 2025 г.

ЦЕЛЬ РАБОТЫ

Изучить критерии выбора стратегий в неан- тагонистической бескоалиционной игре двух игроков на основе равновесия Нэша и оптимальности по Парето. Проверить дан- ные критерии на примере рассмотренных выше игр. Исследовать свойства оптимальных решений неантагонистических бескоалиционных игр на примере биматричных (2×2)-игр.

Задание

а) Сгенерировать случайную биматричную игру (10×10). Найти ситуации, равновесные по Нэшу и оптимальные по Парето, а также пересечение множеств этих ситуаций. Выполнить проверку реализованных алгоритмов на примере трех известных игр: «Семейный спор», «Перекресток», «Дилемма заключенного».

б) Для заданной биматричной (2×2)-игры Г(A, B), пользуясь теоремами о свойствах оптимальных решений, найти ситуации, равновесные по Нэшу, для исходной игры и для ее смешанного расширения.

Исходная матрица, заданная по варианту:

$$\begin{pmatrix} (10,7) & (0,4) \\ (2,1) & (9,3) \end{pmatrix}$$

ХОД РАБОТЫ

Для решения данной лабораторной работы был использован язык программирования Go. В проекте был написан класс для решения бескоалиционных игр.

Пример запуска программы:

```
go run cmd/lw3/main.go
```

Равновесие по Нешу

Ситуацией равновесия по Нэшу называется линия поведения игроков, если она устойчива относительно их индивидуального отклонения, т. е. в этой ситуации ни одному игроку не выгодно изменять своё мнение о выбранной стратегии при сохранении линии поведения другими игроками, так как он первым же от этого и пострадает.

Пусть $x^* = \left(x_1^*,\, x_2^*,\, \ldots,\, x_n^*\right)$ — ситуация (набор стратегий) всех игроков. Говорят, что x^* является *ситуацией равновесия по Нэшу* (в чистых стратегиях), если для каждого игрока $i = 1, 2, \ldots, n$ и любой стратегии $x_i \in X_i$ справедливо неравенство

$$H_i\left(x_i^*, x_{-i}^*\right) \;\geq\; H_i\left(x_i, x_{-i}^*\right),$$

где x_{-i}^* обозначает стратегии всех игроков, кроме игрока i, а H_i — функция выигрыша (или полезности) игрока i.

Совокупность всех равновесных по Нэшу ситуаций игры называется *множеством равновесий Нэша.*

Оптимальность по Парето

Ситуацией, оптимальной по Парето, называется состояние системы, при котором значение каждого частного критерия, описывающего состояние системы, не может быть улучшено без ухудшения положения других элементов.

Рассмотрим множество векторов

$$\{H(x)\} = \{H_1(x), H_2(x), \ldots, H_n(x)\}, \quad x \in X, \quad x = (x_1, x_2, \ldots, x_n),$$

образованных значениями вектор-выигрышей игроков во всех возможных ситуациях $x \in X$.

Ситуация x^* в бескоалиционной игре Γ называется *оптимальной по Парето*, если не существует ситуации $x \in X$, для которой выполняется неравенство

$$H_i(x) \ \geq \ H_i(x^*)$$

хотя бы для одного $i_0 \in N$, и при этом хотя бы для одного игрока $j \in N$ строгое неравенство

$$H_j(x) \ > \ H_j(x^*).$$

1 Случайная биматричная игра

Рассмотрим случайную биматричную игру размером 10x10. Ее решение представлено на рисунке 1 в виде вывода программы в поток вывода.

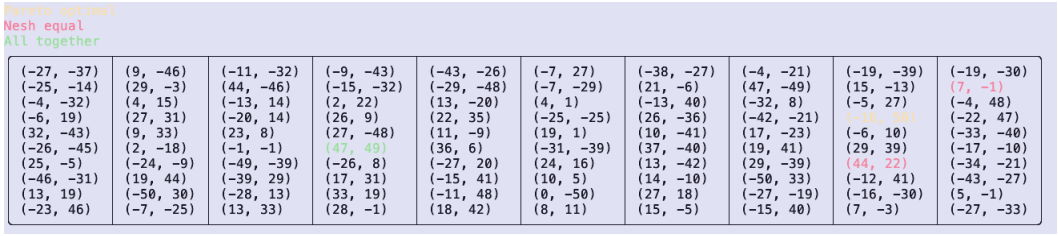

Рисунок 1 – Случайная биматричная игра 10x10

– Ситуации, равновесные по Нешу, покрашены в красный цвет.
– Ситуации, оптимальные по Парето, покрашены в желтый цвет.
– Ситуации, и равновесные по Нэшу, и оптимальные по Парето, покрашены в зеленый цвет.

2 Дилема заключенного

Выполним проверку реализованных алгоритмов на примере известной игры "Дилема заключенного". Вывод программы представлен на рисунке 2.

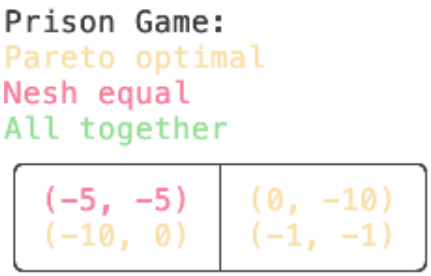

Рисунок 2 – Дилема заключеннго

Описание решения написано в методических материалах к данной лабороторной работе. Ответ, который выдала программа, совпал с тем, что написано в методичке.

3 Семейный спор

Выполним проверку реализованных алгоритмов на примере известной игры "Семейный спор". Вывод программы представлен на рисунке 3.

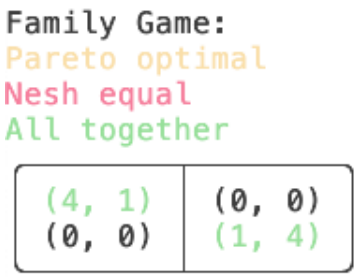

Рисунок 3 – Семейный спор

Описание решения написано в методических материалах к данной лабороторной работе. Ответ, который выдала программа, совпал с тем, что написано в методичке.

4 Перекресток

Выполним проверку реализованных алгоритмов на примере известной игры "Перекресток". Вывод программы представлен на рисунке 4.

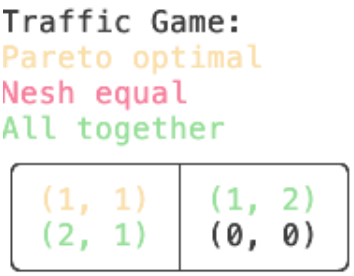

Рисунок 4 – Перекресток

Описание решения написано в методических материалах к данной лабороторной работе. Ответ, который выдала программа, совпал с тем, что написано в методичке.

5 Биматрична игра по варианту

Рассмотрим биматричную игру 2x2, заданную по варианту. Вывод программы представлен на рисунке 5.

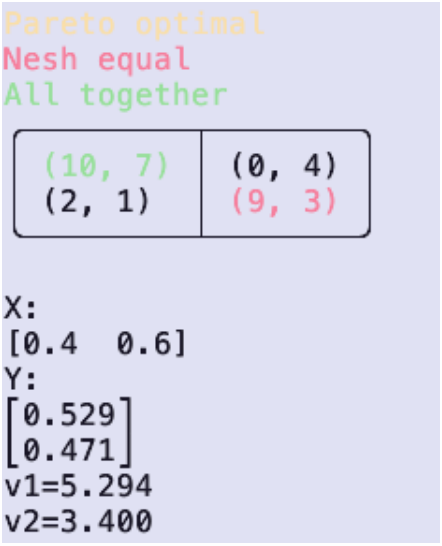

Рисунок 5 – Биматричная игра по варианту 2x2

Как видим, у этой игры есть две равновесные по Нешу систуации в чистых стратегиях, поэтому в смешанном дополнении игры существует еще одна вполне смешанная ситуация равновесия, которую мы можем рассчитать по формулам из теоремы 5.1.

Теорема 5.1. Пусть $\Gamma(A, B)$ — биматричная (2×2)-игра, где A и B являются невырожденными матрицами:

$$(A, B) = \begin{pmatrix} (\alpha_1, \beta_1) & (\alpha_1, \beta_2) \\ (\alpha_2, \beta_1) & (\alpha_2, \beta_2) \end{pmatrix}.$$

Если игра Γ имеет вполне смешанную ситуацию равновесия, то она единственна и вычисляется по формулам:

$$x = v_2\, u\, B^{-1}, \quad y = v_1\, A^{-1} u,$$

где

$$v_1 = \frac{1}{(u\, A^{-1} u)}, \quad v_2 = \frac{1}{(u\, B^{-1} u)}.$$

ЗАКЛЮЧЕНИЕ

В результате выполнения данной лабораторной работы были изучены и проанализированы критерии выбора оптимальных стратегий в неантагонистических бескоалиционных играх. Исследование позволило:

- Детально рассмотреть понятия равновесия по Нэшу и оптимальности по Парето, выделив существенные особенности их определения и интерпретации в контексте игр нескольких игроков.
- Разработать алгоритмы для поиска ситуаций равновесия по Нэшу и оптимальных по Парето решений в биматричных играх, что подтвердилось на примере случайно сгенерированных игр размером 10×10.
- Провести тестирование алгоритмов на известных примерах игр («Дилемма заключенного», «Семейный спор», «Перекресток»), что показало соответствие полученных результатов методическим рекомендациям и теоретическим выкладкам.
- Рассмотреть биматричную игру 2×2, заданную по варианту, и выявить, что наличие нескольких равновесных ситуаций в чистых стратегиях приводит к существованию дополнительного вполне смешанного равновесия в расширении игры, которое можно вычислить по аналитическим формулам.

ПРИЛОЖЕНИЕ А

Класс реализация биматричных игр

Листинг А.1 – bigames.go

```
package bigames

type Position struct {
  X float64
  Y float64

  isPareto bool
  isNesh   bool
}

type BiGame struct {
  m [][]Position
}

func NewBiGame(m [][]Position) *BiGame {
  g := &BiGame{
    m: make([][]Position, len(m)),
  }

  for i := range m {
    g.m[i] = make([]Position, 0, len(m[i]))

    g.m[i] = append(g.m[i], m[i]...)
  }

  return g
}

func (g *BiGame) Solve() *Solution {
  for i, s := range g.m {
    for j := range s {
      if g.isParetoOptimal(i, j) {
        g.m[i][j].isPareto = true
      }

      if g.isNeshEqual(i, j) {
        g.m[i][j].isNesh = true
```

```
      }
    }
  }

  return newSolution(g.m)
}

func (g *BiGame) isParetoOptimal(a, b int) bool {
  for i := range g.m {
    for j := range g.m[i] {
      if g.notParetoCondition(i, j, a, b) {
        return false
      }
    }
  }

  return true
}

func (g *BiGame) notParetoCondition(i, j, a, b int) bool {
  return (g.m[i][j].X >= g.m[a][b].X && g.m[i][j].Y > g.m[a][b].Y)
     ||
    (g.m[i][j].X > g.m[a][b].X && g.m[i][j].Y >= g.m[a][b].Y)
}

func (g *BiGame) isNeshEqual(a, b int) bool {
  for i := range g.m {
    if g.m[i][b].X >= g.m[a][b].X && i != a {
      return false
    }
  }

  for j := range g.m[a] {
    if g.m[a][j].Y >= g.m[a][b].Y && j != b {
      return false
    }
  }

  return true
}
```

Листинг А.2 – solution.go

```
package bigames

import (
  "fmt"
  "strings"

  "github.com/jedib0t/go-pretty/v6/table"
  "gonum.org/v1/gonum/mat"
)

type colorType string

const (
  neshColor    = colorType("\033[31m")
  parettoColor = colorType("\033[33m")
  allColor     = colorType("\033[32m")
  resetColor   = colorType("\033[0m")
)

type Solution struct {
  strB *strings.Builder

  X, Y   *mat.VecDense
  v1, v2 float64

  a, b *mat.Dense
  u    *mat.VecDense
}

func newSolution(m [][]Position) *Solution {
  s := &Solution{
    strB: &strings.Builder{},
  }

  fmt.Fprintf(s.strB, "%sPareto optimal%s\n%sNesh equal%s\n%sAll
     together%s\n",
    parettoColor, resetColor, neshColor, resetColor, allColor,
       resetColor)

  t := table.NewWriter()
```

```
t.SetOutputMirror(s.strB)
t.SetStyle(table.StyleRounded)

aRaw := make([]float64, 0, len(m)*len(m[0]))
bRaw := make([]float64, 0, len(m)*len(m[0]))

for i := range m {
  r := table.Row{}

  for j := range m[i] {
    var color colorType

    if m[i][j].isNesh {
      color = neshColor
    }

    if m[i][j].isPareto {
      color = parettoColor
    }

    if m[i][j].isNesh && m[i][j].isPareto {
      color = allColor
    }

    r = append(r, fmt.Sprintf("%s(%.0f, %.0f)%s",
      color, m[i][j].X, m[i][j].Y, resetColor))

    aRaw = append(aRaw, m[i][j].X)
    bRaw = append(bRaw, m[i][j].Y)
  }

  t.AppendRow(r)
}

t.Render()

u := make([]float64, len(m))
for i := range u {
  u[i] = 1
}

s.a = mat.NewDense(len(m), len(m[0]), aRaw)
```

```
  s.b = mat.NewDense(len(m), len(m[0]), bRaw)
  s.u = mat.NewVecDense(len(m), u)

  return s
}

func (s *Solution) String() string {
  fmt.Fprintf(s.strB, "")

  return s.strB.String()
}

func (s *Solution) SolveMixedEquilibrium() (
  x *mat.Dense, y *mat.VecDense, v1, v2 float64, err error,
) {
  rA, cA := s.a.Dims()
  rB, cB := s.b.Dims()

  fmt.Println(mat.Formatted(s.a))
  fmt.Println(mat.Formatted(s.b))

  if rA != cA || rB != cB || rA != rB {
    return nil, nil, 0, 0, fmt.Errorf("матрицы A и B
       должныбытьодинаковогоразмера    NxN")
  }

  if s.u.Len() != rA {
    return nil, nil, 0, 0, fmt.Errorf("длина вектора u
       должнасовпадатьсразмерамиматриц     A и B")
  }

  var invA, invB mat.Dense

  if err := invA.Inverse(s.a); err != nil {
    return nil, nil, 0, 0, fmt.Errorf("не удалосьобратитьматрицу   A: %
       v", err)
  }

  if err := invB.Inverse(s.b); err != nil {
    return nil, nil, 0, 0, fmt.Errorf("не удалосьобратитьматрицу   B: %
       v", err)
  }
```

```
  Au := mat.NewVecDense(rA, nil)
  Au.MulVec(&invA, s.u)

  var Bu mat.Dense
  Bu.Mul(&invB, s.u)
  fmt.Println(mat.Formatted(&Bu))

  uTAu := mat.Dot(s.u, Au)

  var uTBu mat.Dense
  uTBu.Mul(s.u.T(), &Bu)
  fmt.Println(mat.Formatted(&uTBu))

  v1 = 1.0 / uTAu
  v2 = 1.0 / uTBu.At(0, 0)

  var uInvB mat.Dense
  uInvB.Mul(s.u.T(), &invB)

  var xVec mat.Dense
  xVec.Scale(v2, &uInvB)

  yVec := mat.NewVecDense(rA, nil)
  yVec.ScaleVec(v1, Au)

  return &xVec, yVec, v1, v2, nil
}
```

Листинг А.3 – examples.go

```
package bigames

var (
  e = float64(0.001)

  PrisonGame = [][]Position{
    {{X: -5, Y: -5}, {X: 0, Y: -10}},
    {{X: -10, Y: 0}, {X: -1, Y: -1}},
  }

  FamilyGame = [][]Position{
```

```
    {{X: 4, Y: 1}, {X: 0, Y: 0}},
    {{X: 0, Y: 0}, {X: 1, Y: 4}},
  }

  TrafficGame = [][]Position{
    {{X: 1, Y: 1}, {X: 1 - e, Y: 2}},
    {{X: 2, Y: 1 - e}, {X: 0, Y: 0}},
  }
)
```

ПРИЛОЖЕНИЕ Б

Точка входа в программу

Листинг Б.4 – main.go

```
package main

import (
  "fmt"
  "log"
  "math/rand"

  "github.com/themilchenko/game_theory/internal/bigames"
  "gonum.org/v1/gonum/mat"
)

var PersonalGame = [][]bigames.Position{
  {{X: 10, Y: 7}, {X: 0, Y: 4}},
  {{X: 2, Y: 1}, {X: 9, Y: 3}},
}

func generateRandomGame(n, m int) [][]bigames.Position {
  randGame := make([][]bigames.Position, n)

  for i := range n {
    randGame[i] = make([]bigames.Position, m)

    for j := range m {
      randGame[i][j].X = float64(rand.Intn(101) - 50)
      randGame[i][j].Y = float64(rand.Intn(101) - 50)
    }
  }

  return randGame
}

func main() {
  fmt.Println("Prison Game:")
  fmt.Println(bigames.NewBiGame(bigames.PrisonGame).Solve().String
     ())

  fmt.Println("Family Game:")
```

```
  fmt.Println(bigames.NewBiGame(bigames.FamilyGame).Solve().String
     ())

  fmt.Println("Traffic Game:")
  fmt.Println(bigames.NewBiGame(bigames.TrafficGame).Solve().String
     ())

  fmt.Println("Variant Game:")
  fmt.Println(bigames.NewBiGame(PersonalGame).Solve().String())

  s := bigames.NewBiGame(PersonalGame).Solve()
  x, y, v1, v2, err := s.SolveMixedEquilibrium()
  if err != nil {
    log.Fatal(err)
  }

  fmt.Println(s.String())
  fmt.Printf("X:\n%.3v\nY:\n%.3v\nv1=%.3f\nv2=%.3f\n", mat.
     Formatted(x), mat.Formatted(y),
    v1, v2)

  fmt.Println("Random Game:")
  fmt.Println(bigames.NewBiGame(generateRandomGame(10, 10)).Solve()
     .String())
}
```