Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Институт информационных технологий

Кафедра ИС

# ОТЧЕТ

по лабораторной работе №2

ИССЛЕДОВАНИЕ ПРИМЕНЕНИЯ АППАРАТА ТЕОРИИ ОДНОМЕРНОЙ

ПОЛЕЗНОСТИ ДЛЯ РЕШЕНИЯ ЗАДАЧ ВЫБОРА АЛЬТЕРНАТИВ

Выполнил:

ст. гр. ИС/б-21-2-о

Мовенко К. М.

Проверил:

Кротов К.В.

Севастополь

2024

# ЦЕЛЬ РАБОТЫ

Исследовать применение аппарата теории полезности при принятии решений по выбору альтернатив.

# ЗАДАНИЕ

Вариант 3. Задана матрица отношения нестрогого предпочтения (Рисунок 1). Используя метод, реализующий формирование классов эквивалентности , формирование множества неповторяющихся классов эквивалентности , выполнить разработку программы, определяющей значения функции полезности для этих классов и значения функции для решений , с последующим определением эффективных решений, для которых .

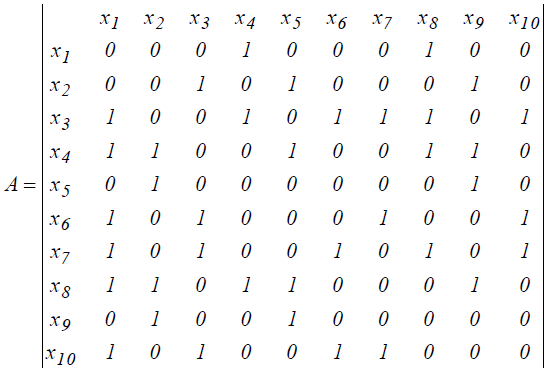


Рисунок 1 – Матрица отношений нестрогого порядка

# ХОД РАБОТЫ

Была написана программа, последовательно выполняющая формирование классов эквивалентности и нахождение значений функции полезности для этих классов. Её правильность была подтверждена на ряде примеров.

Для исходной матрицы не выполняется условие нестрогого упорядочения, однако при и заполнении диагонали единицами (рефлексивность эквивалентности) нахождение полезности возможно.

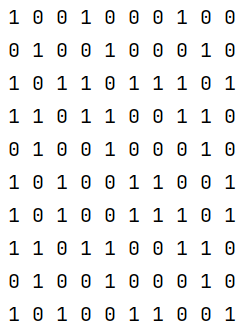


Рисунок 1 – Обновлённая матрица отношений нестрогого порядка

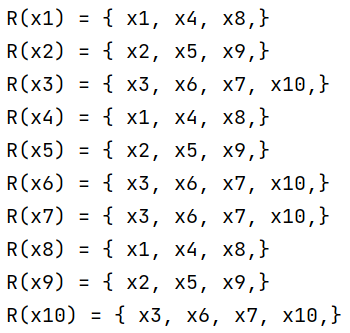


Рисунок 2 – Множества эквивалентных элементов

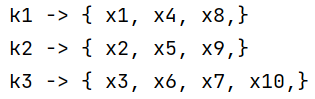


Рисунок 3 – Классы эквивалентности



Рисунок 4 – Матрица предпочтений для классов эквивалентности

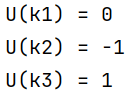


Рисунок 5 – Функция полезности для классов

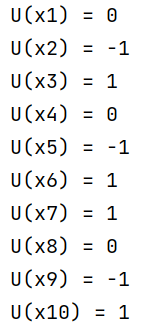


Рисунок 6 – Функция полезности для решений

В данном примере наиболее эффективными решениями являются , , ,.

# ТЕКСТ ПРОГРАММЫ

import copy  
  
  
*# упорядочение по матрице отношений*def is\_collatable(a, n):  
 count = n  
  
 while count > 0:  
 excluded = []  
  
 *# определение исключаемых элементов* for i in range(n):  
 row\_sum = sum(a[i])  
 col\_sum = sum([a[j][i] for j in range(n)])  
 if row\_sum == 0:  
 if col\_sum == 0:  
 return False  
 excluded.append(i)  
  
 if len(excluded) == 0 or len(excluded) == n:  
 return False  
  
 *# обнуление отношений с исключаемыми элементами* for q in range(len(excluded)):  
 for x in range(n):  
 a[x][excluded[q]] = 0  
  
 *# исключение элементов, добавленных в MaxR* for q in range(len(excluded)):  
 for x in range(n):  
 a[excluded[q]][x] = 1  
  
 count -= len(excluded)  
  
 return True  
  
  
*# нахождение значений функции полезности*def findU(l\_prev, h\_prev, seen, m, iter):  
 L = []  
 H = []  
  
 for l in range(0, len(K)):  
 if B[m][l] == 1:  
 if l not in seen:  
 L.append(l)  
 Uk[l] = Uk[m] - 1 *#-iter* iter += 1  
 seen.append(l)  
 else:  
 if h\_prev != -1:  
 Uk[m] = (Uk[l] + Uk[h\_prev]) / 2  
  
 for h in range(0, len(K)):  
 if B[h][m] == 1:  
 if h not in seen:  
 H.append(h)  
 Uk[h] = Uk[m] + 1 *#iter* iter += 1  
 seen.append(h)  
 else:  
 if l\_prev != -1:  
 Uk[m] = (Uk[h] + Uk[l\_prev]) / 2  
  
 for l in L:  
 iter = findU(-1, m, seen, l, iter)  
  
 for h in H:  
 iter = findU(m, -1, seen, h, iter)  
  
 return iter  
  
  
n = 10 *# число альтернатив*A = [  
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],  
 [0, 1, 0, 0, 1, 0, 0, 0, 1, 0],  
 [1, 0, 1, 1, 0, 1, 1, 1, 0, 1],  
 [1, 1, 0, 1, 1, 0, 0, 1, 1, 0],  
 [0, 1, 0, 0, 1, 0, 0, 0, 1, 0],  
 [1, 0, 1, 0, 0, 1, 1, 0, 0, 1],  
 [1, 0, 1, 0, 0, 1, 1, 1, 0, 1],  
 [1, 1, 0, 1, 1, 0, 0, 1, 1, 0],  
 [0, 1, 0, 0, 1, 0, 0, 0, 1, 0],  
 [1, 0, 1, 0, 0, 1, 1, 0, 0, 1]  
]  
  
R = [[] for \_ in range(n)] *# множества экв. элементов*K = [] *# классы эквивалентности*B = [] *# матрица отношений классов экв-ти*Uk = [] *# функция полезности для классов*Ux = [0] \* n *# функция полезности для альтернатив  
  
  
# ВЫЧИСЛЕНИЯ  
  
# заполнение множеств экв. элементов*for i in range(n):  
 for j in range(n):  
 if A[i][j] == A[j][i] == 1:  
 R[i].append(j)  
  
*# идентификация классов эквивалентности*K = list(sorted(set(tuple(k) for k in R)))  
  
*# заполнение матрицы отношений классов эквивалентности*B = [[0] \* len(K) for \_ in range(len(K))]  
for l in range(len(K)):  
 for h in range(len(K)):  
 if l == h:  
 continue  
 summ = sum(A[i][j] for i in K[l] for j in K[h])  
 if summ != 0:  
 B[l][h] = 1  
  
*# проверка возможности упорядочить классы*if not is\_collatable(copy.deepcopy(B), len(K)):  
 print("ОШИБКА: невозможно упорядочить классы")  
 exit()  
  
*# поиск значений U(k) для классов экв-ти*Uk = [0] \* len(K)  
findU(-1, -1, [0], 0, 1)  
  
*# установка значений U(x) для альтернатив*for l in range(len(Uk)):  
 for i in K[l]:  
 Ux[i] = Uk[l]  
  
  
*# ВЫВОД ДАННЫХ*print("Матрица нестрогого предпочтения:")  
for row in A:  
 print(\*row)  
print()  
  
print("Множества эквивалентных элементов:")  
for i in range(n):  
 print("R(x{}) = {{".format(i + 1), end="")  
 for j in R[i]:  
 print(" x{}".format(j + 1), end=",")  
 print("}")  
print()  
  
print("Классы эквивалентности X/~:")  
for l in range(len(K)):  
 print("k{} -> {{".format(l + 1), end="")  
 for i in K[l]:  
 print(" x{}".format(i + 1), end=",")  
 print("}")  
print()  
  
print("Матрица строгого предпочтения классов эквивалентности:")  
for row in B:  
 print(\*row)  
print()  
  
print("Полезность классов эквивалентности:")  
for l in range(len(K)):  
 print("U(k{}) = {}".format(l + 1, Uk[l]))  
print()  
  
print("Полезность альтернатив:")  
for i in range(n):  
 print("U(x{}) = {}".format(i + 1, Ux[i]))  
print()  
  
print("Эффективные решения:")  
print("{", end="")  
for i in range(n):  
 if Ux[i] == max(Ux):  
 print(" x{}".format(i + 1, Ux[i]), end=",")  
print("}")

# ВЫВОД

В ходе работы было исследовано применение аппарата теории полезности при принятии решений по выбору альтернатив.