Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Институт информационных технологий

Кафедра ИС

# ОТЧЕТ

по лабораторной работе №7

ИССЛЕДОВАНИЕ МЕТОДОВ ГРУППОВОГО ВЫБОРА РЕШЕНИЙ

Выполнил:

ст. гр. ИС/б-21-2-о

Мовенко К. М.

Проверил:

Кротов К.В.

Севастополь

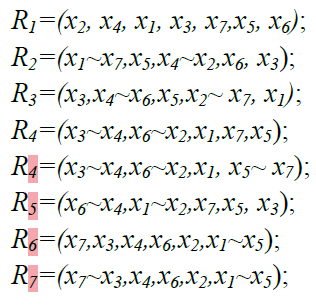
2024

# ЦЕЛЬ РАБОТЫ

Исследовать способы определения эффективных решений при групповом выборе.

# ЗАДАНИЕ

Выполнить определение итогового ранжирования R для 8 исходных ранжирований следующего вида:



# ХОД РАБОТЫ

Частные ранжирования были представлены в виде матриц сравнений:

По матрицам сравнений были определены значения матрицы потерь . Для этого для каждой пары индексов матрицы были выполнены расчёты, подобные представленным ниже.

По ней была построена матрица потерь .

Минимальное значение матрицы – 202 (). Альтернативы в итоговом ранжировании были расставлены так: .

Матрицы были модифицированы путём исключения из них 4-го столбца и 5-ой строки.

Аналогичным образом были выполнены ещё две итерации алгоритма. В итоге было получено следующее итоговое ранжирование:

# ТЕКСТ ПРОГРАММЫ

Листинг 1 – Код программы

import numpy as np  
  
  
*# вывод матрицы*def print\_matrix(a):  
 for row in a:  
 print(' '.join('%2i' % x for x in row))  
 print()  
  
  
*# нахождение коэффициента потерь*def find\_p(A, i, j, index):  
 p = 0  
  
 for k in range(len(A)):  
 if k == index:  
 continue  
 p += abs(A[index][i][j] - A[k][i][j])  
  
 return int(p)  
  
  
*# вывод итогового ранжирования*def print\_R(Rl, Rr, end=False):  
 print('R = { ', end='')  
  
 Rl = sorted(Rl.items(), key=lambda item: item[1])  
 Rr = sorted(Rr.items(), key=lambda item: item[1])  
  
 for item in Rl:  
 print('x%i' % (item[0] + 1), end=' ')  
 if end == False:  
 print('... ', end='')  
 for item in Rr:  
 print('x%i' % (item[0] + 1), end=' ')  
 print('}\n')  
  
  
*# итоговое ранжирование через алгоритм Кемени*def Kemeni(A, cols\_i, rows\_i, Rl={}, Rr={}, iter=1):  
 n = A.shape[1] *# число решений* m = A.shape[0] *# число ранжирований  
  
 # проверка окончания для нечётного числа решений* count = []  
 for x in set(cols\_i + rows\_i):  
 if x not in Rl.keys() and x not in Rr.keys():  
 count.append(x)  
 if len(count) == 1:  
 max\_key = max(Rl.values())  
 Rl[count[0]] = max\_key + 1  
  
 print('Оставшееся решение - x%i\n' % (count[0]+1))  
 return Rl, Rr  
  
 *# расчёт матрицы потерь* P = np.zeros((n, n))  
 for i in range(n):  
 for j in range(n):  
 r = [A[k][i][j] for k in range(m)]  
 r\_max\_i = np.argmax(r)  
  
 P[i][j] = find\_p(A, i, j, r\_max\_i)  
  
 *# определение матрицы Q* E = np.ones((n, n)) - np.eye(n)  
 Q = np.dot(np.dot(E, P), E)  
  
 min\_val = np.inf  
 min\_index = None  
  
 *# поиск qmin* for index in np.ndindex(Q.shape):  
 i = rows\_i[index[0]]  
 j = cols\_i[index[1]]  
  
 if (all(x not in Rl.keys() and x not in Rr.keys() for x in (i, j))  
 and i != j and Q[index] < min\_val):  
 min\_val = Q[index]  
 min\_index = index  
  
 *# проверка окончания для чётного числа решений* if not min\_index:  
 print('')  
 return Rl, Rr  
  
  
 *# ВЫВОД ВЫЧИСЛЕНИЙ В ИТЕРАЦИИ* print('-' \* 5 + ' Итерация №%i ' % iter + '-' \* 5, end='\n\n')  
  
 for k in range(m):  
 print('Ранжирование R%i:' % (k + 1))  
 print\_matrix(A[k])  
  
 print('Матрица потерь P:')  
 print\_matrix(P)  
  
 print('Матрица Q:')  
 print\_matrix(Q)  
  
 i = rows\_i[min\_index[0]]  
 j = cols\_i[min\_index[1]]  
  
 print('qmin = %i' % min\_val)  
  
 *# размещение альтернатив в итоговое ранжирование* if not Rl:  
 Rl[j] = 0  
 Rr[i] = n-1  
 else:  
 max\_key = max(Rl.values())  
 min\_key = min(Rr.values())  
 Rl[j] = max\_key + 1  
 Rr[i] = min\_key - 1  
  
 print\_R(Rl, Rr)  
  
 *# удаление лишних столбцов и строк* new\_A = np.empty((0, n-1, n-1))  
 for k in range(m):  
 a = np.delete(A[k], min\_index[0], axis=0)  
 a = np.delete(a, min\_index[1], axis=1)  
 new\_A = np.concatenate((new\_A, a[np.newaxis, :, :]), axis=0)  
  
 cols\_i.remove(j)  
 rows\_i.remove(i)  
  
 return Kemeni(new\_A, cols\_i, rows\_i, Rl, Rr, iter+1)  
  
  
*# исходные ранжирования*R = [  
 [np.array([2, 4, 1, 3, 7, 5, 6]), []],  
 [np.array([1, 7, 5, 4, 2, 6, 3]), [(1, 7), (4, 2)]],  
 [np.array([3, 4, 6, 5, 2, 7, 1]), [(4, 6), (2, 7)]],  
 [np.array([3, 4, 6, 2, 1, 7, 5]), [(3, 4), (6, 2)]],  
 [np.array([3, 4, 6, 2, 1, 5, 7]), [(3, 4), (6, 2), (5, 7)]],  
 [np.array([6, 4, 1, 2, 7, 5, 3]), [(6, 4), (1, 2)]],  
 [np.array([7, 3, 4, 6, 2, 1, 5]), [(1, 5)]],  
 [np.array([7, 3, 4, 6, 2, 1, 5]), [(7, 3), (1, 5)]]  
]A = []  
n = len(R[0][0]) *# число решений  
  
# построение матрицы для каждого ранжирования*for k in range(len(R)):  
 Rk = R[k]  
  
 Ak = np.zeros((n, n))  
  
 for h in range(n-1):  
 for l in range(h+1, n):  
 i = Rk[0][h] - 1  
 j = Rk[0][l] - 1  
  
 Ak[i][j] = 1  
 Ak[j][i] = -1  
  
 for eq in Rk[1]:  
 i = eq[0] - 1  
 j = eq[1] - 1  
  
 Ak[i][j] = 0  
 Ak[j][i] = 0  
  
 A.append(Ak)  
  
Rl, Rr = Kemeni(np.array(A), list(range(n)), list(range(n)))  
print('Итоговое ранжирование:')  
print\_R(Rl, Rr, end=True)

# ВЫВОД

В ходе работы были исследованы способы определения эффективных решений при групповом выборе.