

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

по дисциплине "Системный анализ и принятие решений"

Выполнил
студент гр. в3530904/00030

В.С. Баганов

Руководитель
доцент, к.т.н.

В.В. Амосов

«_____» _____ 202__ г.

Санкт-Петербург
2023

Содержание

1. Введение	3
2. Математическая модель системы поддержки принятия решения	4
2.1. Механизм доминирования по БО (R):	4
2.2. Механизм блокировки по БО (R):	4
2.3. Турнирный механизм	4
2.4. Механизм К-максимальных	5
2.5. Выбор лучшего варианта проводится по всем ранжированиям:	7
3. Программная реализация СППР	8
3.1. Программная реализация СППР в виде веб-приложения, демонстрация её работы.	8
3.2. Подготовка данных для СППР	8
3.3. Архитектурный уровень разработки приложения	11
4. Использование СППР при выборе машины.	13
4.1. Вывод работы программы	16
5. Использование СППР при выборе робота-пылесоса	18
5.1. Механизм доминирования	20
5.2. Механизм блокировки	21
5.3. Турнирный механизм	21
5.4. Определение k-максимальных элементов	21
5.5. Вывод работы программы	22
5.6. Выводы	23
5.7. Приложение. Промежуточные матричные вычисления.	25
6. Листинг программы Decision_system.cpp	28

1. Введение

Экзаменационная работа посвящена разработке, программной реализации в виде веб-приложения и тестированию на примерах системы поддержки принятия решения (СППР) на основе качественного подхода к принятию решения (ПР).

С необходимостью принятия решения сталкиваются и при разработке искусственного интеллекта, и при машинном обучении, и при разработке систем реального времени.

Принятие решения может быть осуществлено несколькими способами, как предлагаемым, так и на основе количественного подхода.

Качественный подход к принятию решения в данной статье подразумевает формализацию ситуации ПР с использованием аппарата бинарных отношений (БО) или предпочтений, задание для каждого БО весового коэффициента, проведение выбора решений с помощью аппарата функций выбора (ФВ), поиск оптимальных вариантов решений для каждого БО и сведения полученных результатов в обобщающую таблицу для наглядности и автоматизации выбора лицом, принимающим решение (ЛПР). Возможно автоматическое принятие решения.

Система поддержки принятия решений (СППР) (англ. Decision Support System, DSS) — компьютерная автоматизированная система, целью которой является помощь людям, принимающим решение в сложных условиях для полного и объективного анализа предметной деятельности. Это означает, что она выдаёт информацию (в печатной форме, или на экране монитора, или звуком), основываясь на входных данных, помогающую людям быстро и точно оценить ситуацию и принять решение. СППР возникли в результате слияния управленческих информационных систем и систем управления базами данных.

Для анализа и выработок предложений в СППР используются разные методы. Это могут быть: информационный поиск, интеллектуальный анализ данных, поиск знаний в базах данных, рассуждение на основе прецедентов, имитационное моделирование, эволюционные вычисления и генетические алгоритмы, нейронные сети, ситуационный анализ, когнитивное моделирование и других. Некоторые из этих методов были разработаны в рамках искусственного интеллекта. Если в основе работы СППР лежат методы искусственного интеллекта, то говорят об интеллектуализированной СППР или ИСППР. Близкие к СППР классы систем — это экспертные системы и автоматизированные системы управления.

2. Математическая модель системы поддержки принятия решения

Математическая модель системы поддержки принятия решений, описание методов её реализации в программе.

Бинарные отношения задаются матрицами вручную с помощью экспертов, либо с помощью отдельной программы.

Аппарат функций выбора задаётся для каждого БО механизмами доминирования, блокировки, турнирным механизмом и механизмом определения К-максимальных вариантов [1]. Для каждого БО задаются весовые коэффициенты (κ). Полученные по каждому механизму результаты ранжируются с учётом весовых коэффициентов БО.

По механизмам доминирования и блокировки определяется сколько раз варианты решения доминировали и блокировали по разным БО, затем для каждого доминирующего или блокирующего по БО варианта формируем сумму из весовых коэффициентов соответствующих БО, и исходя из этого варианты решения ранжируются по убыванию с учётом весовых коэффициентов БО.

2.1. Механизм доминирования по БО (R):

Выбираются те альтернативы, из которых идут стрелки R ко всем остальным альтернативам.

В матричном представлении из множества альтернатив выбираются те, для которых каждая конкретная матрица критериев содержит в строках все единицы, кроме возможно той, которая соответствует данной альтернативе. Таким образом, получается список частных решений. Чтобы получить общее решение, выбирается такая альтернатива, которая входит во все частные.

$$C_R(x) = \{x \in X \mid \forall y \in X, xRy\}$$

2.2. Механизм блокировки по БО (R):

Выбор «не улучшаемых» по R элементов, т.е. лучше которых нет.

В матричном представлении из множества альтернатив выбираются те, для которых каждая конкретная матрица критериев содержит в столбцах все единицы, кроме возможно той, которая соответствует данной альтернативе. Таким образом, получается список частных решений. Чтобы получить общее решение, выбирается такая альтернатива, которая входит во все частные.

$$C_R(x) = \{x \in X \mid \forall y \in X, x\bar{R}y\}$$

Результаты работы турнирного механизма по каждому БО умножаются на соответствующие этим БО весовые коэффициенты (κ), при этом для каждого варианта в каждом БО получаем своё произведение, затем для каждого варианта решения вычисляем сумму из этих произведений. Исходя из полученных сумм, ранжируем варианты решения по убыванию.

2.3. Турнирный механизм

Механизм используется для ранжирования всех альтернатив.

Частные решения для каждой матрицы критериев получаются из условия: чем больше сумма для данной альтернативы, тем данная альтернатива лучше подходит в качестве решения. Для получения общего решения используется следующий механизм: для каждой альтернативы находится сумма по критериям, где слагаемые равны значению турнирной функции по данному критерию, умноженному на значимость данного критерия. Среди полученных значений выбирается наибольшее, второе по величине, третье и т.д., это и определяет то место, которое займет данная альтернатива в общем решении.

Для каждого БО (R) рассматриваем все варианты X (x1, ..., xn) и для каждого варианта x определяем сумму, перебирая остальные варианты y

$$f_R(x) = \sum f_R(x, y)$$

где

$$f_R(x, y) = \begin{cases} 1, & (xRy) \wedge (x\bar{R}y) \\ 0, & (xRy) \wedge (x\bar{R}y) \\ 1/2, & \end{cases}$$

y - остальные варианты решения $\in X$,

X – предъявленное множество вариантов,

x и y – элементы множества вариантов X,

$f_R(x, y) - xy$

$f_R(x) - x, f_R(x)$,

в каждом БО перемножаем эти числа $f_R(x) * (f_R(x))$,

для каждого варианта x получаем сумму произведений ($\kappa^*(f_R(x))$),

по полученным суммам ранжируем варианты, вариант с наибольшей суммой будет лучшим по турнирному механизму.

Поиск оптимальных вариантов решений состоит в поиске k-максимальных вариантов ($\kappa=1,2,3,4$) для каждого БО и проверке их на оптимальность.

2.4. Механизм K-максимальных

Механизм поиска наилучшего решения с использованием k-максимальных вариантов решения реализован следующим образом: сначала для каждого решения определяется, является ли оно 4-ым максимальным, 3-им максимальным, 2-ым максимальным, 1-ым максимальным, затем вычисляется общее значение максимальности альтернативы, например, если альтернатива 4, 3, 1-максимальна, то общее значение будет равно 8. Затем общее значение умножается на значимость критерия и происходит суммирование по всем критериям для данной альтернативы, чем больше полученная сумма, тем ближе альтернатива к лучшей.

Перебираем для каждого БО все варианты x (x1, ..., xn) и для каждого варианта x: определяем количество вариантов, подчинённых x по каждому БО R

$$H_R^0(x) = \{ y \in X, y \neq x \mid xRy \wedge y\bar{R}x \}$$

определяем количество вариантов, эквивалентных x по каждому БО R

$$E_R(x) = \{ y \in X, y \neq x \mid xRy \wedge yRx \}$$

определяем количество вариантов, несравнимых с x по каждому БО R

$$N_R(x) = \{y \in X, y \neq x \mid x\bar{R}y \wedge y\bar{R}x\}$$

определяем количество вариантов, подчинённых, эквивалентных и несравнимых x по каждому БО R

$$S_R^1(x) = H_R^0(x) \cup E_R(x) \cup N_R(x)$$

определяем количество вариантов, подчинённых и несравнимых x по каждому БО R

$$S_R^2(x) = H_R^0(x) \cup N_R(x)$$

определяем количество вариантов подчинённых и эквивалентных x по каждому БО R

$$S_R^3(x) = H_R^0(x) \cup E_R(x)$$

определяем количество вариантов подчинённых x по каждому БО R

$$S_R^4(x) = H_R^0(x)$$

для каждого БО перебираем все варианты x (x_1, \dots, x_n) и для каждого варианта x формируем вектор из 4 компонентов

$$X_1 : (S_R^1(x_1), S_R^2(x_1), S_R^3(x_1), S_R^4(x_1))$$

$$X_2 : (S_R^1(x_2), S_R^2(x_2), S_R^3(x_2), S_R^4(x_2))$$

.....

$$X_n : (S_R^1(x_n), S_R^2(x_n), S_R^3(x_n), S_R^4(x_n))$$

или в развернутом виде

$$X_1 : H_R^0(x_1) + E_R(x_1) + N_R(x_1), H_R^0(x_1) + N_R(x_1), H_R^0(x_1) + E_R(x_1), H_R^0(x_1)$$

$$X_2 : H_R^0(x_2) + E_R(x_2) + N_R(x_2), H_R^0(x_2) + N_R(x_2), H_R^0(x_2) + E_R(x_2), H_R^0(x_2)$$

.....

$$X_n : H_R^0(x_n) + E_R(x_n) + N_R(x_n), H_R^0(x_n) + N_R(x_n), H_R^0(x_n) + E_R(x_n), H_R^0(x_n)$$

где n – количество вариантов решения.

По каждому варианту от 1 до n для каждого БО получили числа $S_{Ri}(Xj), i = 1, \dots, 4, j = 1, \dots, n$.

Затем в рамках каждого отдельного БО для каждого варианта (x_j) получаем сумму по i от 1 до 4 $\sum S_{Ri}(Xj)$

и эту сумму умножаем на соответствующий этому БО весовой коэффициент «к», то есть

$$S_j = \sum S_{Ri}(Xj), i = 1, \dots, 4.$$

Получаем для каждого БО столбец из $S_j, j = 1, \dots, n$.

Определяем для каждого варианта « j » сумму из S_j по каждому БО:

$$S_{jp} = \sum S_{ij}, i = 1, \dots, p,$$

где p -количество БО.

Получаем для всех БО один столбец сумм $S_j p, .$

Для каждого БО по каждому компоненту векторов всех вариантов по 1-му, 2- му, 3-му и 4-му компоненту находим максимальные значения!

Им будут соответствовать варианты, которые будут являться, соответственно, 1-максимальным, 2-максимальным, 3-максимальным и 4-максимальным вариантами!

После сравниваем численные значения 1,2,3,4-максимальных вариантов с общим количеством вариантов (n) и определяем являются ли эти k -максимальные варианты ещё и оптимальными вариантами (максимальными, строго максимальными, наибольшими и строго наибольшими). В результате находим оптимальные варианты для каждого БО.

Найденные оптимальные варианты также ранжируются с учётом весовых коэффициентов БО. Для этого определяем сколько раз (число «М») в разных БО найденные варианты были оптимальными: максимальными, строго максимальными, наибольшими и строго наибольшими. В разных БО для оптимальных вариантов имеем разные значения $S_j . S_j m 1 M : S_j M = \sum S_j m,$

но только сумму тех $S_j, .$

Получаем для всех БО столбец сумм $S_j M, .$

Таким образом для механизма K -максимальных вариантов имеем два ранжирования: по суммам $S_j p S_j M$

2.5. Выбор лучшего варианта проводится по всем ранжированиям:

по механизмам доминирования и блокировки, по турнирному механизму и по механизму определения K -максимальных вариантов, основываясь на балльной системе: балл («В») для каждого варианта при каждом механизме определяется как количество вариантов « n » плюс 1 минус номер места в ранжированном столбце (« L_j »), $B_j = n + 1 - L_j.$

3. Программная реализация СППР

3.1. Программная реализация СППР в виде веб-приложения, демонстрация её работы.

На рисунке 3.1 представлен системный уровень разработки СППР, в котором приложение представлено чёрным ящиком с описанием входных и выходных данных.



Рисунок 3.1. Системный уровень разработки.

3.2. Подготовка данных для СППР

На вход программы подается 3 файла:

optionsForComparison.txt

(Массив вариантов)

В начале файла идут два числа

Первое число - количество параметров, по которым будет проходить сравнения.

Второе - количество сравниваемых элементов. Далее идёт сам массив данных.

```
1 4 3
2 13000 150000 148000
3 2008 2009 2009
4 170000 140000 150000
5 60 70 80
```

weightCoefficients.txt

(Массив данных сил параметров)

В файле должно быть столько же элементов, сколько и строк в основном массиве.

Сумма всех элементов должна быть равна 1.

```
1 0.3
2 0.3
```



```
3 0.1
4 0.3
```

dataComparisonConf.txt

(Массив видов сравнений)

В файле должно быть столько же элементов, сколько и строк в основном массиве:

0 - А лучше, когда $A > B$ (пример: год выпуска)

1 - А лучше, когда $A < B$ (пример: цена)

-1 - Подсчёт не идёт

```
1 1
2 0
3 1
4 0
```

На выходе получаем вывод итоговых данных в консоль и промежуточные вычисления в файлы (по каждому элементу сравнения).

Вывод в консоль:

```
Консоль отладки Microsoft Visual Studio
-----Входящие данные-----
13000 15000 148000
2008 2009 2009
170000 140000 150000
60 70 80
-----Механизм доминирования-----
1 0.3 3
2 0.4 2
2 0.6 1
-----Механизм блокировки-----
1 0.3 1
1 0.1 2
1 0.3 1
-----Турнирный механизм-----
0.6 3
0.95 2
1.45 1
-----Механизм k-Max вариантов-----
2.4 3 4.8 1
3.8 2 1.6 2
5.8 1 4.8 1
-----Механизмы (бальная система)-----
1 3 1 1 3 9 3
2 2 2 2 2 10 2
3 3 3 3 3 15 1

C:\Users\bagano\source\repos\SAPR2\Debug\SAPR2.exe (процесс 2552) заведе
Чтобы автоматически закрывать консоль при остановке отладки, включите
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

После того, как система считала данные из файлов, она на основании массива вариантов и массива сигнала сравнений выстраивает матрицу сравнения пар вариантов для каждого предпочтения. Матрица сравнения пар вариантов представляет из себя двумерный массив в котором каждый столбец является результатом сравнением двух чисел, а каждая строка обозначает число. В каждый момент сравнения напротив большего значения ставится 1, а наименьшего -1. Напротив чисел, которые не участвуют в сравнении, ставится 0.

К примеру, в первом столбце можно наблюдать сравнение первого элемента со вторым во втором столбце сравнение первого и третьего элемента и в третьем столбце сравнение второго и третьего элемента. Если в массиве вариантов встречаются два равных элемента, то в этом случае система обработает это как два сравнения с весом 0.5

Далее к этой матрице применяются требуемые механизмы и результаты их работы записываются в матрицу результатов. Где в столбцы с первый по третий записывается результат Механизма Доминирования, с четвертого по шестой результат Механизма Блокировки, в седьмой и восьмой результат Турнирного Механизма и столбцы с 9 и 11 содержат в себе информацию по результату Механизма К-максимальных.

out0.txt

```

1 This is: 1 element
2 1      1      0
3 -1     0     -1
4 0      -1     1

```

out1.txt

```

1 This is: 2 element
2 -1      0     -1      0
3 1      0.5   0     -0.5
4 0     -0.5   1      0.5

```

out2.txt

```

1 This is: 3 element
2 -1      0     -1
3 1      1      0
4 0     -1      1

```

out3.txt

```

1 This is: 4 element
2 -1     -1      0
3 1      0     -1
4 0      1      1

```

3.3. Архитектурный уровень разработки приложения

Архитектурный уровень разработки представлен на рисунке 3.2. В нём приложение СППР описывается структурной блок-схемой.

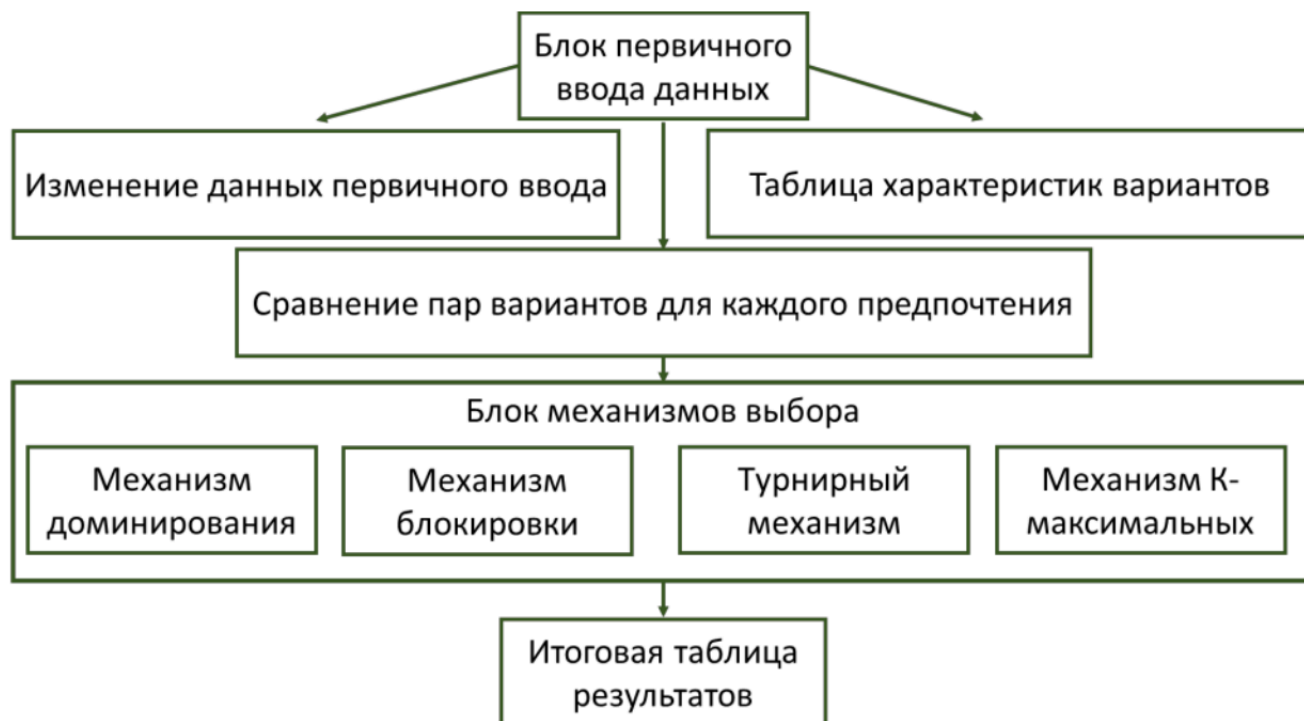


Рисунок 3.2. Архитектурный уровень разработки.

- Тестирование приложения будет выполняться в ручном режиме.
- Для тестирования необходимо разработать вторую версию приложения.

На рисунке 3.3 представлены две версии приложения: версия разработчика и пользовательская версия.

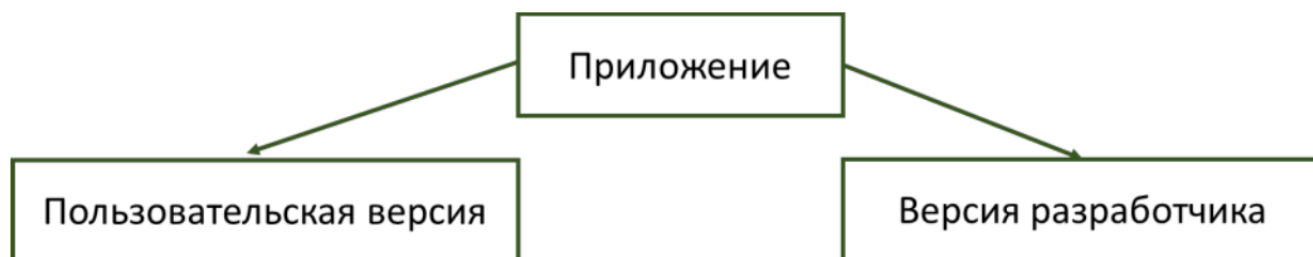


Рисунок 3.3. Версии приложения

Версия разработчика:

- Реализует полный функционал пользовательской версии, имея при этом дополнительные возможности.

- Предусмотрены дополнительные экранные управляющие кнопки, вызывающие выполнение операций, автоматически выполняемых в пользовательской версии, что позволяет разработчику контролировать правильность обработки данных в любой операции.
- Предусмотрен вывод в консоль браузера промежуточных данных, получаемых на различных этапах выполнения программы.

4. Использование СППР при выборе машины.

Для демонстрации приложения произведём выбор, приняв для сравнения три автомобиля одной модели. Данные взяты из сайта <http://www.auto.ru> и представлены в таблице на рисунке 4.

LADA (BA3) Priora I, 2008 (1вариант)	LADA (BA3) Priora I, 2009 (2 вариант)	LADA (BA3) Priora I, 2009 (3 вариант)
143000 руб.	150000 руб.	148000 руб.
Пробег 170000 км	Пробег 140000 км	Пробег 150000 км
1.6 л / 98 л. с. / Бензин	1.6 л / 81 л. с. / Бензин	1.6 л / 81 л. с. / Бензин
Механическая	Механическая	Механическая
Серый, Передний, Седан	Серый, Передний, Седан	Белый, Передний, Хэтчбек 5 дверей

Рисунок 4.1. Данные

Данные о предпочтениях и их весовых коэффициентах заполняются в таблице с экранными управляющими кнопками, представленной на рис 5.

На рисунке 6 приведена форма представления предпочтений вариантов в виде таблицы, заполняемая в процессе работы приложения.

Таблица предпочтений вариантов			
	Цена	Год выпуска	Пробег
LADA Priora 1	143000	2008	170000
LADA Priora 2	150000	2009	140000
LADA Priora 3	148000	2009	150000
Создать таблицу	Ввести данные таблицы		Удалить таблицу

Рисунок 4.2. Данные

Результаты работы механизмов доминирования, блокировки, турнирного и К- максимальных вариантов с учётом весовых коэффициентов предпочтений представлены в таблице на рисунках 7, 8, 9, 10.

Механизм доминирования			
Доминирующие варианты по каждому предпочтению	Количество доминирующих позиций у вариантов	Баллы вариантов с учётом весовых коэффициентов	Место L_j
Цена: LADA Priora 1 Год выпуска: LADA Priora2, LADA Priora 3 Пробег: LADA Priora2	LADA Priora 1: 1	LADA Priora 1: 0.3	2
	LADA Priora 2: 2	LADA Priora 2: 0.7	1
	LADA Priora 3: 1	LADA Priora 3: 0.3	2

Рисунок 4.3. Данные

Здесь при расчёте итогового количества баллов в рамках механизма доминирования каждая доминирующая позиция добавляет к результату варианта количество баллов равное весовому коэффициенту предпочтения, в котором получена данная доминирующая позиция, затем определяются места вариантов L_j .

Механизм блокировки			
Блокирующие варианты по предпочтениям	Количество блокирующих позиций у вариантов	Баллы вариантов с учётом весовых коэффициентов	Место L_j
Цена: LADA Priora 1 Пробег: LADA Priora2	LADA Priora 1: 1	LADA Priora 1: 0.3	2
	LADA Priora 2: 1	LADA Priora 2: 0.4	1
	LADA Priora 3: 0	LADA Priora 3: 0	3

Рисунок 4.4. Данные

Здесь при расчёте итогового количества баллов в рамках механизма блокировки каждая блокирующая позиция добавляет к результату варианта количество баллов равное весовому коэффициенту предпочтения, в котором получена данная блокирующая позиция, затем определяются места вариантов L_j .

Турнирный механизм	
Баллы вариантов с учётом весовых коэффициентов	Место L_j
LADA Priora 1: 0.60	3
LADA Priora 2: 1.25	1
LADA Priora 3: 1.15	2

Рисунок 4.5. Данные

Здесь итоговая позиция варианта определяется количеством баллов в рамках турнирного механизма, полученным вариантом непосредственно в механизме, который учитывает весовые коэффициенты, затем определяются места вариантов L_j .

Механизм K-таx вариантов				
Варианты	Суммы S_{jp}	Место L_j	Суммы S_{jM}	Место L_j
LADA Priora 1	2.4	3	4.8	2
LADA Priora 2	5.0	1	6.4	1
LADA Priora 3	4.6	2	0	3

Рисунок 4.6. Данные

Здесь итоговая позиция варианта определяется в результате ранжирования: по суммам $S_{jp}S_{jM}, L_j$. — 11.

Варианты	Механизмы (балльная система)					Сумма баллов (В)
	Блоки- ровки	Домини- рования	Турнир- ный	K-max (S _{jp})	K-max (S _{jM})	
LADA Priora 1	2	2	1	1	2	8
LADA Priora 2	3	3	3	3	3	15
LADA Priora 3	1	2	2	2	1	8
Вывести результаты выбора						

Рисунок 4.7. Данные

Здесь итоговая позиция варианта определяется по балльной системе. Балл («В») для каждого варианта при каждом механизме определяется как количество вариантов «п» плюс 1 минус номер места в ранжированном столбце («L_j»), $B_j = n + 1 - L_j$.

В результате работы СППР выбирается второй вариант LADA Priora 2: LADA (BA3) Priora I, 2009 за 150000 рублей. Он набрал наибольшее количество баллов: 15 и вышел на 1 место. Интерес представляют 2 и 3 варианты, они набрали одинаковое количество баллов по 8. Если убрать ранжирование по суммам S_{jp} по всем вариантам, а оставить ранжирование только по оптимальным вариантам, то на итоговое 2 место выйдет 1 вариант, если наоборот, то 3 вариант. Разрешению конфликта может помочь введение дополнительного предпочтения, например по мощности двигателя, тогда 1 вариант будет на 2 месте опередив 3 вариант.

4.1. Вывод работы программы

Для тестирования программы были добавлены дополнительные параметры мощности машин (60,70,80) для оптимального распределения победителей. По итогам по сумме баллов победил 3 вариант (LADA (BA3) PRIORA I(3 вариант)).

-----Входящие данные-----

```
13000 150000 148000
2008 2009 2009
170000 140000 150000
60 70 80
```

-----Механизм доминирования-----

```
1 0.3 3
2 0.4 2
2 0.6 1
```

-----Механизм блокировки-----

```
1 0.3 1
1 0.1 2
1 0.3 1
```

-----Турнирный механизм-----

```
0.6 3
0.95 2
1.45 1
```

-----Механизм k-Max вариантов-----

```
2.4 3 4.8 1
3.8 2 1.6 2
5.8 1 4.8 1
```

-----Механизмы (бальная система)-----

```
1 3 1 1 3 9 3
2 2 2 2 2 10 2
3 3 3 3 3 15 1
```

C:\Users\bagano\source\repos\SAPR2\Debug\SAPR2.exe (процесс 2552) завершил работу.
 Чтобы автоматически закрывать консоль при остановке отладки, включите "Отображать панель отладки".
 Чтобы автоматически закрыть консоль при остановке отладки".
 Нажмите любую клавишу, чтобы закрыть это окно...

5. Использование СППР при выборе робота-пылесоса


Для демонстрации приложения было выбрано 7 моделей роботов-пылесосов с относительно близкими ценами и параметрами. Данные взяты из сайта Яндекс Маркет: [Ссылка на популярные роботы-пылесосы](#)

Ниже приведен рисунок с пылесосами и бальной системой каждой отдельной модели самого Яндекс Маркета (выделены зелеными флажками). Для проверки правильности работы нашей СППР, сравним результаты бальной системы Яндекс Макета и СППР.

Сравнение товаров

Мобильные телефоны 5 **Роботы-пылесосы 7** Лодочные моторы 1 Ноутбуки 1


Показывать: Различающиеся характеристики Все характеристики Удалить список



Робот-пылесос Xiaomi Mi Robot Vacuum Cleaner 1S

4,8 1209 отзывов


от 17 190 ₽
30 предложений



Робот-пылесос Kitfort KT-545

4,6 153 отзыва


от 14 989 ₽
11 предложений



Робот-пылесос Eufy RoboVac 35C

4,8 68 отзывов


от 12 648 ₽
8 предложений



Робот-пылесос dreame F9

4,8 1224 отзыва


от 16 177 ₽
78 предложений



Робот-пылесос Xiaomi Mi Robot Vacuum-Mop Essential

4,7 1941 отзыв


от 16 230 ₽
35 предложений



Робот-пылесос iRobot Roomba 698

4,8 332 отзыва

от 15 460 ₽
22 предложения



Робот-пылесос Xiaomi Mi Robot Vacuum-Mop 2 Lite

4,3 14 отзывов

от 15 669 ₽
74 предложения

Общие характеристики ^

ТИП УБОРКИ ?

сухая сухая и влажная сухая сухая и влажная сухая и влажная сухая сухая и влажная

ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ ?

постоянная карты постоянная карты программирование по постоянная карты постоянная карты программирование по постоянная карты

Параметры всех пылесосов были сведены в таблицу, чтобы определить важные для принятия решения параметры.

	Робот-пылесос						
	Xiaomi Mi Robot Vacuum Cleaner 1S	Kitfort KT-545	Eufy RoboVac 35C	dreame F9	Mi Robot Vacuum-Mop Essential	iRobot Roomba 698	Xiaomi Mi Robot Vacuum-Mop 2 Lite
Критерии сравнения							
Рейтинг покупателей	4,80	4,60	4,80	4,80	4,70	4,80	4,30
Отзывы	1209	153	68	1224	1941	332	14
Тип уборки (0 -сухая, 1 - и влажная)	0	1	0	1	1	0	1
Время работы от аккумулятора	150	100	100	150	90	180	100
Тип аккумулятора	1	1	1	1	1	1	1
Емкость аккумулятора (мА·ч)	5 200	2 500	2 600	5200	2 500	1 800	2600
Объем контейнера для пыли	0,42	0,6	0,6	0,6	0,6	0,6	0,45
Тип контейнера	0	1	0	1	1	0	1
Мощность всасывания (Вт)	25	25	40	25	25	25	25
Габариты (высота пылесоса в см)	9,6	7,4	7,25	8	8,2	9,4	8,13
Цена	17 190	14 989	12 648	16 177	16 230	15 460	15 669
Номер в программе	1	2	3	4	5	6	7

optionsForComparison.txt

(Массив вариантов)

В начале файла идут два числа

Первое число - количество параметров, по которым будет проходить сравнения.

Второе - количество сравниваемых элементов. Далее идёт сам массив данных.

```

1 10 7
2 1209 153 68 1224 1941 332 14
3 0 1 0 1 1 0 1
4 150 100 100 150 90 180 100
5 1 1 1 1 1 1 1
6 5200 2500 2600 5200 2500 1800 2600
7 0.42 0.6 0.6 0.6 0.6 0.6 0.45
8 0 1 0 1 1 0 1
9 25 25 40 25 25 25 25
10 9.6 7.4 7.25 8 8.2 9.4 8.13
11 17190 14989 12648 16177 16230 15460 15669

```

Ниже представлена таблица с весовыми коэффициентами и сигналами сравнения. Самые важные критерии:

- Отзывы (говорит о долговечности работы в различных условиях и простоте использования.)
- Тип уборки сухая/влажная (больше функций, чище воздух)
- Время работы от аккумулятора (большая площадь охвата уборки квартиры, 1- или 3-х комнатная кв.)
- Мощность всасывания (чем больше мощность, тем громче работа, и быстрее происходит уборка. Выбор в зависимости от целей.)

Так как все пылесосы примерно в одной ценовой категории, поэтому упор был сделан на качественные характеристики пылесосов.

Критерии сравнения	Весовые коэффициенты/Значимость	Предпочтения/Сигналы сравнения
Рейтинг покупателей		
Отзывы	0,14	0
Тип уборки (0 -сухая, 1 - и влажная)	0,14	0
Время работы от аккумулятора	0,15	0
Тип аккумулятора	0,04	1
Емкость аккумулятора (мА·ч)	0,11	0
Объем контейнера для пыли	0,09	0
Тип контейнера	0,05	0
Мощность всасывания (Вт)	0,14	0
Габариты (высота пылесоса в см)	0,09	0
Цена	0,05	1

Подготовка данных для загрузки в СППР.
weightCoefficients.txt

```

1 0,14
2 0,14
3 0,15
4 0,04
5 0,11
6 0,09
7 0,05
8 0,14
9 0,09
10 0,05

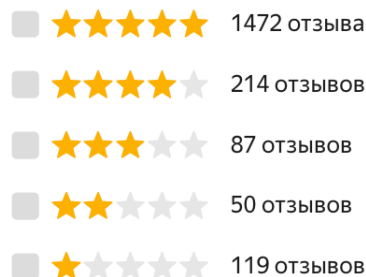
```

dataComparisonConf.txt

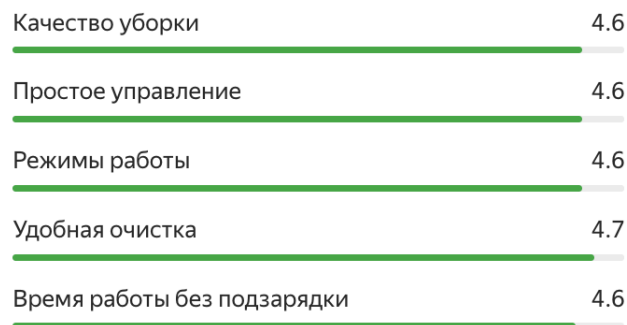
1	0
2	0
3	0
4	1
5	0
6	0
7	0
8	0
9	0
10	1

Кстати Яндекс Маркет использует свои критерии сравнения для бальной системы по всем товарам.

Отзывы с оценкой



90% покупателей
рекомендуют этот товар



5.1. Механизм доминирования

Выбираются те альтернативы, из которых идут стрелки R ко всем остальным альтернативам.

В матричном представлении из множества альтернатив выбираются те, для которых каждая конкретная матрица критериев содержит в строках все единицы, кроме возможно той, которая соответствует данной альтернативе. Таким образом, получается список частных решений. Чтобы получить общее решение, выбирается такая альтернатива, которая входит во все частные.

5.2. Механизм блокировки

Выбор «не улучшаемых» по R элементов, т.е. лучше которых нет.

В матричном представлении из множества альтернатив выбираются те, для которых каждая конкретная матрица критериев содержит в столбцах все единицы, кроме возможно той, которая соответствует данной альтернативе. Таким образом, получается список частных решений. Чтобы получить общее решение, выбирается такая альтернатива, которая входит во все частные.

5.3. Турнирный механизм

Механизм используется для ранжирования всех альтернатив.

Частные решения для каждой матрицы критериев получаются из условия: чем больше сумма для данной альтернативы, тем данная альтернатива лучше подходит в качестве решения. Для получения общего решения используется следующий механизм: для каждой альтернативы находится сумма по критериям, где слагаемые равны значению турнирной функции по данному критерию, умноженному на значимость данного критерия. Среди полученных значений выбирается наибольшее, второе по величине, третье и т.д., это и определяет то место, которое займет данная альтернатива в общем решении.

5.4. Определение k-максимальных элементов

Механизм поиска наилучшего решения с использованием k-максимальных вариантов решения реализован следующим образом: сначала для каждого решения определяется, является ли оно 7-ым максимальным и т.д., 3-им максимальным, 2-ым максимальным, 1-ым максимальным, затем вычисляется общее значение максимальнойности альтернативы, например, если альтернатива 4, 3, 1-максимальна, то общее значение будет равно 8. Затем общее значение умножается на значимость критерия и происходит суммирование по всем критериям для данной альтернативы, чем больше полученная сумма, тем ближе альтернатива к лучшей.

5.5. Вывод работы программы

```

cs Консоль отладки Microsoft Visual Studio
-----Входящие данные-----
1209    153    68    1224    1941    332    14
0       1      0      1      1      0      1
150     100    100    150    90     180    100
1       1      1      1      1      1      1
5200    2500   2600   5200   2500   1800   2600
0.42    0.6    0.6    0.6    0.6    0.6    0.45
0       1      0      1      1      0      1
25      25     40     25     25     25     25
9.6     7.4    7.25   8      8.2    9.4    8.13
17190   14989   12648   16177   16230   15460   15669
-----Механизм доминирования-----
3       0.24    5
4       0.32    3
4       0.32    3
5       0.43    2
5       0.46    1
3       0.28    4
3       0.23    6
-----Механизм блокировки-----
1       0.09    4
0       0      5
2       0.19    1
0       0      5
1       0.14    3
1       0.15    2
0       0      5
-----Турнирный механизм-----
3.04    3
2.77    5
2.635   6
3.945   1
3.1     2
2.99    4
2.52    7
-----Механизм k-Мак вариантов-----
12.16   3      4.32    4
11.08   5      0      5
10.54   6      9.12    1
15.78   1      0      5
12.4    2      6.72    3
11.96   4      7.2     2
10.08   7      0      5
-----Механизмы (бальная система)-----
0       0      1      1      0      2      5
1       0      0      0      0      1      6
1       3      0      0      3      7      3
2       0      3      3      0      8      2
3       1      2      2      1      9      1
0       2      0      0      2      4      4
0       0      0      0      0      0      7

```

5.6. Выводы

В тройку лучших попали пылесосы под номерами 3,4,5.

	Робот-пылесос						
Критерии сравнения	Xiaomi Mi Robot Vacuum Cleaner 1S	Kitfort KT-545	Eufy RoboVac 35C	dreame F9	Mi Robot Vacuum-Mop Essential	iRobot Roomba 698	Xiaomi Mi Robot Vacuum-Mop 2 Lite
Рейтинг покупателей	4,80	4,60	4,80	4,80	4,70	4,80	4,30
Отзывы	1209	153	68	1224	1941	332	14
Тип уборки (0 -сухая, 1 - и влажная)	0	1	0	1	1	0	1
Время работы от аккумулятора	150	100	100	150	90	180	100
Тип аккумулятора	1	1	1	1	1	1	1
Емкость аккумулятора (мА·ч)	5 200	2 500	2 600	5200	2 500	1 800	2600
Объем контейнера для пыли	0,42	0,6	0,6	0,6	0,6	0,6	0,45
Тип контейнера	0	1	0	1	1	0	1
Мощность всасывания (Вт)	25	25	40	25	25	25	25
Габариты (высота пылесоса в см)	9,6	7,4	7,25	8	8,2	9,4	8,13
Цена	17 190	14 989	12 648	16 177	16 230	15 460	15 669
Номер в программе	1	2	3	4	5	6	7



Робот-пылесос Eufy RoboVac 35C

4.8

68 отзывов

от 12 648 ₽

9 предложений



Робот-пылесос dreame F9

4.8

1226 отзывов

от 16 139 ₽

81 предложение



Робот-пылесос Xiaomi Mi Robot Vacuum-Mop Essential

4.7

1942 отзыва

от 16 890 ₽

39 предложений

При работе СППР (при заданных весах и предпочтениях в том числе низкой стоимости) победитель под номером 5. Это доказывают и отзывы покупателей. В первую тройку победителей попали роботы-пылесосы с максимальными рейтингами самого Яндекс Маркета (см. зеленые флажки на рисунке). Так же хотел бы добавить, что пользуюсь этим пылесосом и он превосходит все ожидания, но чтобы сделать правильный выбор в ручном режиме пришлось потратить гораздо больше времени (2 недели, чтобы посмотреть видео обзоры), чем СППР.



Робот-пылесос Xiaomi Mi Robot Vacuum-Mop Essential

4.7

1942 отзыва

от 16 890 ₽

39 предложений

Отзывы с оценкой

- ★★★★★ 1474 отзыва
- ★★★★☆ 214 отзывов
- ★★★★☆ 87 отзывов
- ★★★☆☆ 50 отзывов
- ★★☆☆☆ 119 отзывов

90% покупателей
рекомендуют этот товар

Качество уборки	4.6
Простое управление	4.6
Режимы работы	4.6
Удобная очистка	4.7
Время работы без подзарядки	4.6

5.7. Приложение. Промежуточные матричные вычисления.

```

1 This is: 1 element
2 1 1 1 1 0 0 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0
3 -1 0 0 0 1 1 0 0 -1 0 0 0 0 -1 0 0 0 0 -1 0 0
4 0 -1 0 0 -1 0 1 0 0 -1 0 0 0 0 -1 0 0 0 0 -1 0
5 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 -1 0 0 0 0 0
6 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
7 0 0 -1 0 0 0 0 0 0 0 -1 0 0 0 0 0 -1 0 1 1 1
8 0 0 0 -1 0 -1 -1 0 0 0 0 -1 0 0 0 0 0 -1 0 0 -1

```

```

1 This is: 2 element
2 0.5 0.5 -1 0 0 0 0 0 -0.5 0 -1 0 0 0 0 0 -1 0 0 0 0
  ↳ 0 -0.5 0 -1 0 0 0 0 0 0
3 0 0 1 1 0.5 0.5 1 0.5 0 0 0 -0.5 0 0 0 0 0 -0.5 0 0 0
  ↳ 0 0 0 0 -0.5 0 0 0 0
4 -0.5 0 0 -1 0 0 0 0 0.5 0.5 0 0 -1 0 0 0 0 0 -1 0 0
  ↳ 0 0 -0.5 0 0 -1 0 0 0
5 0 0 0 0 -0.5 0 0 0 0 0 1 0.5 1 0.5 1 0.5 0 0 0 -0.5 0
  ↳ 0 0 0 0 0 -0.5 0 0
6 0 0 0 0 0 -0.5 0 0 0 0 0 0 -0.5 0 0 1 0.5 1 0.5 1
  ↳ 0.5 0 0 0 0 0 -0.5 0
7 0 -0.5 0 0 0 0 -1 0 0 -0.5 0 0 0 0 -1 0 0 0 0 -1 0
  ↳ 0.5 0.5 0 0 0 0 0 -1
8 0 0 0 0 0 0 0 -0.5 0 0 0 0 0 0 -0.5 0 0 0 0 -0.5
  ↳ 0 0 1 0.5 1 0.5 0.5 1

```

```

1 This is: 3 element
2 1 1 0.5 1 1 0 0 0 0 0 0 -0.5 0 0 0 0 -1 0 0 0 0 0
  ↳ 0 0
3 -1 0 0 0 0 0.5 1 0.5 -0.5 0 0 0 -1 0 0 0 0 -1 0 0 0
  ↳ 0 -0.5 0 0
4 0 -1 0 0 0 -0.5 0 0 0.5 1 0.5 0 0 -1 0 0 0 0 -1 0 0
  ↳ 0 0 -0.5 0
5 0 0 -0.5 0 0 0 0 0 0 0 0.5 1 1 1 1 0 0 0 -1 0 0 0
  ↳ 0 0
6 0 0 0 -1 0 0 -1 0 0 -1 0 0 0 0 -1 0 0 0 0 -1 0 0
  ↳ 0 -1
7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
8 0 0 0 0 -1 0 0 -0.5 0 0 -0.5 0 0 0 0 -1 0 0 0 0 0 -1
  ↳ 0.5 0.5 1

```

```

1 This is: 4 element
2 0.5 0.5 0.5 0.5 0.5 0.5 -0.5 0 0 0 0 0 -0.5 0 0 0 0 0
  ↳ -0.5 0 0 0 0 0 -0.5 0 0 0 0 0 -0.5 0 0 0 0 -0.5
  ↳ 0 0 0 0 0
3 -0.5 0 0 0 0 0 0.5 0.5 0.5 0.5 0.5 0.5 0 -0.5 0 0 0 0 0
  ↳ -0.5 0 0 0 0 0 -0.5 0 0 0 0 0 -0.5 0 0 0 0 -0.5
  ↳ 0 0 0 0
4 0 -0.5 0 0 0 0 0 -0.5 0 0 0 0.5 0.5 0.5 0.5 0.5 0.5 0
  ↳ 0 -0.5 0 0 0 0 0 -0.5 0 0 0 0 -0.5 0 0 0 0 0
  ↳ -0.5 0 0 0

```

```

5  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0.5  0.5
   ↳  0.5  0.5  0.5  0.5  0  0  0  0  -0.5  0  0  0  0  0  0  -0.5  0  0  0  0  0
   ↳ -0.5  0  0
6  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0
   ↳ -0.5  0  0  0.5  0.5  0.5  0.5  0.5  0.5  0  0  0  0  0  -0.5  0  0  0
   ↳  0  0  -0.5  0
7  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0
   ↳ -0.5  0  0  0  0  0  -0.5  0  0.5  0.5  0.5  0.5  0.5  0.5  0  0  0
   ↳  0  0  -0.5
8  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0
   ↳  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0.5  0.5  0.5  0.5
   ↳  0.5  0.5

```

```

1  This is: 5 element
2  1  1  0.5  1  1  1  0  0  0  0  0  0  -0.5  0  0  0  0  0  0  0  0  0
   ↳  0
3  -1  0  0  0  0  0  0.5  1  -1  0  0  0  0  -1  0  0  0  0  -0.5  0  -1  0
   ↳  0  0
4  0  -1  0  0  0  0  0  0  1  1  1  0.5  0  0  -1  0  0  0  0  0  0  -0.5
   ↳  0  0
5  0  0  -0.5  0  0  0  0  0  0  0  0  0  0.5  1  1  1  1  1  0  0  0  0
   ↳  0
6  0  0  0  -1  0  0  -0.5  0  0  -1  0  0  0  0  0  -1  0  0  0.5  1  0  0
   ↳ -1  0
7  0  0  0  0  -1  0  0  -1  0  0  -1  0  0  0  0  0  -1  0  0  -1  0  0
   ↳ -1
8  0  0  0  0  0  -1  0  0  0  0  0  0  -0.5  0  0  0  0  0  -1  0  0  1  0.5
   ↳  1  1

```

```

1  This is: 6 element
2  -1  0  0  0  0  0  -1  0  0  0  0  0  -1  0  0  0  0  0  -1  0  0  0
   ↳  0  -1  0  0  0  0  -1
3  1  0.5  0.5  0.5  0.5  1  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0
   ↳ -0.5  0  0  0  0  0  -0.5  0  0  0  0
4  0  -0.5  0  0  0  0  1  0.5  0.5  0.5  0.5  1  0  0  -0.5  0  0  0  0  0
   ↳ -0.5  0  0  0  0  0  -0.5  0  0  0  0
5  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  1  0.5  0.5  0.5  0.5  1  0  0
   ↳  0  -0.5  0  0  0  0  -0.5  0  0  0
6  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  1  0.5
   ↳  0.5  0.5  0.5  1  0  0  0  0  -0.5  0  0
7  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0  0
   ↳ -0.5  0  1  0.5  0.5  0.5  0.5  1  0
8  0  0  0  0  0  -1  0  0  0  0  0  -1  0  0  0  0  0  -1  0  0  0  0
   ↳ -1  0  0  0  0  0  -1  1

```

```

1  This is: 7 element
2  0.5  0.5  -1  0  0  0  0  -0.5  0  -1  0  0  0  0  0  -1  0  0  0  0
   ↳  0  -0.5  0  -1  0  0  0  0
3  0  0  1  1  0.5  0.5  1  0.5  0  0  0  -0.5  0  0  0  0  0  -0.5  0  0  0
   ↳  0  0  0  0  -0.5  0  0  0
4  -0.5  0  0  -1  0  0  0  0.5  0.5  0  0  -1  0  0  0  0  0  -1  0  0
   ↳  0  0  -0.5  0  0  -1  0  0

```

```

5  0  0  0  0  -0.5  0  0  0  0  0  1  0.5  1  0.5  1  0.5  0  0  0  -0.5  0
   ↳ 0  0  0  0  0  0  0  -0.5  0  0
6  0  0  0  0  0  -0.5  0  0  0  0  0  0  0  -0.5  0  0  1  0.5  1  0.5  1
   ↳ 0.5  0  0  0  0  0  0  0  -0.5  0
7  0  -0.5  0  0  0  0  -1  0  0  -0.5  0  0  0  0  -1  0  0  0  0  0  -1  0
   ↳ 0.5  0.5  0  0  0  0  0  0  -1
8  0  0  0  0  0  0  0  -0.5  0  0  0  0  0  0  0  -0.5  0  0  0  0  0  -0.5
   ↳ 0  0  1  0.5  1  0.5  0.5  1

```

```

1  This is: 8 element
2  0.5  0.5  0.5  0.5  0.5  -0.5  0  0  0  0  -1  0  0  0  0  0  -0.5  0  0
   ↳ 0  0  -0.5  0  0  0  0  -0.5  0  0  0  0  -0.5  0  0  0  0
3  -0.5  0  0  0  0  0.5  0.5  0.5  0.5  0.5  0  -1  0  0  0  0  0  -0.5  0
   ↳ 0  0  0  -0.5  0  0  0  0  -0.5  0  0  0  0  -0.5  0  0  0
4  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  0  0  0  0  0  0  0  0
   ↳ 0  0  0  0  0  0  0  0  0  0  0  0
5  0  -0.5  0  0  0  0  -0.5  0  0  0  0  0  -1  0  0  0  0.5  0.5  0.5  0.5
   ↳ 0.5  0  0  -0.5  0  0  0  0  -0.5  0  0  0  0  -0.5  0  0
6  0  0  -0.5  0  0  0  0  -0.5  0  0  0  0  0  -1  0  0  0  0  -0.5  0  0
   ↳ 0.5  0.5  0.5  0.5  0.5  0  0  0  -0.5  0  0  0  0  -0.5  0
7  0  0  0  -0.5  0  0  0  0  -0.5  0  0  0  0  0  -1  0  0  0  0  -0.5  0
   ↳ 0  0  0  -0.5  0  0.5  0.5  0.5  0.5  0.5  0  0  0  0  -0.5
8  0  0  0  0  -0.5  0  0  0  0  -0.5  0  0  0  0  0  -1  0  0  0  0  -0.5
   ↳ 0  0  0  0  -0.5  0  0  0  0  -0.5  0.5  0.5  0.5  0.5  0.5

```

```

1  This is: 9 element
2  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
3  -1  0  0  0  0  0  1  -1  0  -1  0  0  0  -1  0  0  0  -1  0  0
4  0  -1  0  0  0  0  -1  0  -1  0  -1  0  0  0  -1  0  0  0  -1  0
5  0  0  -1  0  0  0  0  1  1  0  0  -1  0  0  0  -1  0  0  0  -1
6  0  0  0  -1  0  0  0  0  0  1  1  1  1  0  0  0  -1  0  0  0  0
7  0  0  0  0  -1  0  0  0  0  0  0  0  1  1  1  1  1  0  0  0  0
8  0  0  0  0  0  -1  0  0  0  0  0  0  -1  0  0  0  0  -1  1  1  1

```

```

1  This is: 10 element
2  -1  0  0  0  0  0  -1  0  0  0  0  0  -1  0  -1  -1  0  0  0  -1  0  0
3  1  1  1  1  1  0  -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
4  0  0  0  0  0  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0
5  0  -1  0  0  0  0  0  -1  0  0  0  1  1  0  0  -1  0  0  0  -1  0
6  0  0  -1  0  0  0  0  0  -1  0  0  0  -1  1  0  0  -1  0  0  0  -1
7  0  0  0  -1  0  0  0  0  0  -1  0  0  0  1  1  1  1  0  0  0  0
8  0  0  0  0  -1  0  0  0  0  0  -1  0  0  0  0  0  0  -1  1  1  1

```

6. Листинг программы Decision_system.cpp

```
1  #include <iostream>
2  #include <stdlib.h>
3  #include <fstream>
4  #include <Windows.h>
5  #include <string>
6  #include <map>
7
8  const char PATH_TO_MAIN_DATA[] =
9  ↪ "IN/vacuum_cleaner_comparison/optionsForComparison.txt";
10 const char PATH_TO_INPUT_COMP[] =
11 ↪ "IN/vacuum_cleaner_comparison/dataComparisonConf.txt";
12 const char PATH_TO_INPUT_POWER[] =
13 ↪ "IN/vacuum_cleaner_comparison/weightCoefficient.txt";
14
15 //Decision_data.txt
16 //Основной массив данных
17 //Вначале идут два числа указывающие на количество элементов, потом сам
18 ↪ массив
19 //Первое число в массиве – количество строк АКА количество параметров по
20 ↪ которым идут сравнения
21 //Второе – количество столбцов АКА количество сравниваемых элементов
22
23 //Data_comparison_conf.txt
24 //массив видов сравнений
25 //В файле должно быть столько же элементов, сколько и строк в основном
26 ↪ массиве
27 //0 – А круче, когда  $A > B$  (пример: год выпуска)
28 //1 – А круче, когда  $A < B$  (пример: цена)
29 //-1 – Подсчёт не идёт
30
31 //Data_power.txt
32 //Массив данных сил параметров
33 //В файле должно быть столько же элементов, сколько и строк в основном
34 ↪ массиве
35 //Сумма всех элементов должна быть равна
36
37 //считать из файла первых два числа (используется что бы понять какой
38 ↪ размер у двумерного массива, который будет считаться)
39 std::pair<double, double> sizeArr(std::ifstream& in)
40 {
41     double i, j;
42     in >> i >> j;
43     return std::make_pair(i, j);
44 }
45
46 //выделение памяти для динамического двухмерного массива
47 double** createArr(int n, int m)
48 {
49     double** A;
50     A = new double* [n];
51     for (int i = 0; i < n; i++)
52     {
53         A[i] = new double[m];
54     }
55     return A;
56 }
```

```

50
51 //вывести двумерный массив
52 void writeArr(int starti, int startj, double** arr, double n, double m)
53 {
54     for (int i = starti; i < n; i++)
55     {
56         for (int j = startj; j < m; j++)
57         {
58             std::cout << arr[i][j] << "\t";
59         }
60         std::cout << std::endl;
61     }
62 }
63
64 //вывести одномерный массив
65 void writeArr_x1(double* arr, double n)
66 {
67     for (int i = 0; i < n; i++)
68     {
69         std::cout << arr[i] << "\t";
70     }
71     std::cout << std::endl;
72 }
73
74
75 //считать массив размером NxM из входящего потока
76 void readFile(double** arr, double n, double m, std::ifstream& in)
77 {
78     for (int i = 0; i < n; i++)
79     {
80         for (int j = 0; j < m; j++)
81         {
82             in >> arr[i][j];
83         }
84     }
85 }
86
87 //уничтожить двумерный массив
88 void distractionArray(double** arr, int n)
89 {
90     for (int i = 0; i < n; ++i)
91     {
92         delete[] arr[i];
93     }
94     delete[] arr;
95 }
96
97 //Вывести элемент пары на экран
98 void show(std::pair<double, double> i) //Функция, которая будет
    ↳ передаваться в алгоритм
99 {
100     std::cout << i.first << " "; //просто выводит параметр на экран
101 }
102
103 //Записать массив в файл
104 void outputmf(double** a, double n, double m, int numofrow,
    ↳ std::ofstream& f)
105 {
106     int i, j;

```

```

107     std::string strname = "OUT/out_dom_" + std::to_string(numofrow) +
    ↪     ".txt";
108     f.open(strname);
109     if (!f)
110     {
111         std::cerr << "File can not be opened\n";
112         exit(0);
113     }
114
115     f << "This is: " << numofrow + 1 << " element" << std::endl;
116
117     for (i = 0; i < n; i++)
118     {
119         for (j = 0; j < m; j++)
120             f << a[i][j] << "\t";
121         f << std::endl;
122     }
123     f.close();
124     f.clear();
125 }
126
127 //Построить матрицу на основе бинарных отношений (двунаправленный граф
    ↪     отражённый на матрице)
128 double** dom_matrix_creat_0(double* Dics_data, double** Dom_data, int n,
    ↪     int count) {
129
130     for (int i = 0; i < n; i++)
131     {
132         for (int j = 0; j < count; j++)
133         {
134             Dom_data[i][j] = 0;
135         }
136     }
137
138     int count_row = 0;
139
140     for (int i = 0; i < n; i++)
141     {
142         for (int j = 0; j < n; j++)
143         {
144             if ((Dics_data[i] == Dics_data[j]) and (i != j))
145             {
146                 Dom_data[i][count_row] = 0.5;
147                 Dom_data[j][count_row] = -0.5;
148                 count_row++;
149             }
150             else if ((Dics_data[i] > Dics_data[j]) and (i != j))
151             {
152                 Dom_data[i][count_row] = 1;
153                 Dom_data[j][count_row] = -1;
154                 count_row++;
155             }
156         }
157     }
158
159     return Dom_data;
160 }
161
162 double** dom_matrix_creat_1(const double* Dics_data, double** Dom_data,
    ↪     int n, int count) {

```

```

163
164     for (int i = 0; i < n; i++)
165     {
166         for (int j = 0; j < count; j++)
167         {
168             Dom_data[i][j] = 0;
169         }
170     }
171
172     int count_row = 0;
173
174     for (int i = 0; i < n; i++)
175     {
176         for (int j = 0; j < n; j++)
177         {
178             if ((Dics_data[i] == Dics_data[j]) and (i != j))
179             {
180                 Dom_data[i][count_row] = 0.5;
181                 Dom_data[j][count_row] = -0.5;
182                 count_row++;
183             }
184             else if ((Dics_data[i] < Dics_data[j]) and (i != j))
185             {
186                 Dom_data[i][count_row] = 1;
187                 Dom_data[j][count_row] = -1;
188                 count_row++;
189             }
190         }
191     }
192
193     return Dom_data;
194 }
195
196 //получить количество связей из строки матрицы бинарных отношений
197 int map_count_con(const double* Dics_data, int n)
198 {
199     int count_row = 0;
200
201     for (int i = 0; i < n; i++)
202     {
203         for (int j = 0; j < n; j++)
204         {
205             if ((Dics_data[i] ≥ Dics_data[j]) and (i != j))
206             {
207                 count_row++;
208             }
209         }
210     }
211
212     return count_row;
213 }
214
215 //почитать призовое место в матрице
216 int count_max_pow(double* Dics_data, int n, double si)
217 {
218     std::map<double, double> mp;
219
220     int count = 1;
221
222     for (int i = 0; i < n; i++) {

```

```

223         mp[Dics_data[i]] = i;
224     }
225
226     for (auto& item : mp)
227     {
228         if (item.first > si)
229             count++;
230     }
231     return count;
232 }
233
234 //почитать призовое место в матрице
235 int count_point(double si)
236 {
237     if (si == 1)
238         return 3;
239     else if (si == 2)
240         return 2;
241     else if (si == 3)
242         return 1;
243     else
244         return 0;
245 }
246
247 //посчитать количество доминирующих связей в строке матрице
248 ↪ (двунаправленный граф отражённый на матрице)
249 int count_dom_in_row(const double* Dics_data, int n) {
250
251     int dom_count = 0;
252
253     for (int j = 0; j < n; j++)
254     {
255         if ((Dics_data[j] == 1) or (Dics_data[j] == 0.5))
256         {
257             dom_count++;
258         }
259     }
260
261     return dom_count;
262 }
263
264 //посчитать силу
265 double count_dom_pow_in_row(const double* Dics_data, int n) {
266
267     double dom_count = 0.0;
268
269     for (int j = 0; j < n; j++)
270     {
271         if (Dics_data[j] > 0)
272         {
273             dom_count += Dics_data[j];
274         }
275     }
276     return dom_count;
277 }
278
279 int block_in_row(const double* Dics_data, int n) {
280
281

```



```

282     int dom_count = 0;
283
284     for (int j = 0; j < n; j++)
285     {
286         if ((Dics_data[j] == -1) or (Dics_data[j] == -0.5))
287         {
288             dom_count++;
289         }
290     }
291
292     if (dom_count != 0)
293         return 0;
294     else
295         return 1;
296 }
297
298 int count_clear_dom(const double* Dics_data, int n) {
299
300     int dom_count = 0;
301
302     for (int j = 0; j < n; j++)
303     {
304         if ((Dics_data[j] == 1))
305         {
306             dom_count++;
307         }
308     }
309
310     return dom_count;
311 }
312
313 int count_ecv_dom(const double* Dics_data, int n) {
314
315     int dom_count = 0;
316
317     for (int j = 0; j < n; j++)
318     {
319         if ((Dics_data[j] == 0.5))
320         {
321             dom_count++;
322         }
323     }
324
325     return dom_count;
326 }
327
328 void writeArrTheBest(double** arr, double n, double m, int numRows)
329 {
330     HANDLE hConsole;
331
332     hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
333     CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
334
335     SetConsoleTextAttribute(hConsole, 8);
336     //SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
337     ↪ csbiInfo.wAttributes);
338
339
340     for (int i = 0; i < n; i++)

```

```

341 {
342     if (arr[i][6] == 1)
343     {
344         SetConsoleTextAttribute(hConsole, 2);
345     }
346     else if (arr[i][6] == 2)
347     {
348         SetConsoleTextAttribute(hConsole, 3);
349     }
350     else if (arr[i][6] == 3)
351     {
352         SetConsoleTextAttribute(hConsole, 1);
353     }
354     for (int j = 0; j < m; j++)
355     {
356         std::cout << arr[i][j] << "\t";
357     }
358     std::cout << std::endl;
359     SetConsoleTextAttribute(hConsole, 7);
360 }
361 }
362 }
363
364
365 int main()
366 {
367     std::ofstream f; //поток для записи данных в файл
368     std::ifstream in; //поток для считывания основных данных
369     std::ifstream in_pow; //поток для считывания данных о силе критерия
370     std::ifstream in_com; //поток для считывания данных о том какой вид
371     ↪ сравнения надо использоваться
372     std::ifstream in_nam; //поток для считывания данных о том какой вид
373     ↪ сравнения надо использоваться
374
375     int x;
376     double errTemp;
377     int Result_data_size = 14;
378     int kmax_result_size = 4;
379
380     in.open(PATH_TO_MAIN_DATA);
381     in_pow.open(PATH_TO_INPUT_POWER);
382     in_com.open(PATH_TO_INPUT_COMP);
383
384     std::pair<int, int> addres = sizeArr(in);
385     double** Dics_data = createArr(addres.first, addres.second);
386     readFile(Dics_data, addres.first, addres.second, in);
387
388     //Массив сравнений
389     x = 0;
390     auto* Comp_arr = new double[addres.first];
391     while (!in_com.eof()) {
392         in_com >> Comp_arr[x];
393         x++;
394     }
395     if (x != addres.first) {
396         std::cout <<
397         ↪ "Error Comp: the number of elements in Data_comparison_conf.txt does not

```

```

398 // массив силы!
399 x = 0;
400 errTemp = 0.0;
401 auto* Pow_arr = new double[adres.first];
402 while (!in_pow.eof()) {
403     in_pow >> Pow_arr[x];
404     errTemp += Pow_arr[x];
405     x++;
406 }
407 if (x != adres.first) {
408     std::cout <<
409         ↪ "Error Pow: the number of elements in Data_power.txt does not match the
410     return 0;
411 }
412
413 in.close();
414 in_pow.close();
415 in_com.close();
416
417 double** Result_data = createArr(adres.second, Result_data_size);
418 for (int i = 0; i < adres.second; i++)
419 {
420     for (int j = 0; j < Result_data_size; j++)
421     {
422         Result_data[i][j] = 0;
423     }
424 }
425
426 for (int i = 0; i < adres.first; i++)
427 {
428     int count = map_count_con(Dics_data[i], adres.second);
429     double** Dom_data = createArr(adres.second, count);
430     double** kMax = createArr(adres.second, kmax_result_size);
431
432     auto* arr = new double[adres.second];
433     double Mull_koef = 0;
434
435     //создание графа отношений
436     if (Comp_arr[i] == 0.0)
437         Dom_data = dom_matrix_creat_0(Dics_data[i], Dom_data,
438         ↪ adres.second, count);
439     else if (Comp_arr[i] == 1.0)
440         Dom_data = dom_matrix_creat_1(Dics_data[i], Dom_data,
441         ↪ adres.second, count);
442     else if (Comp_arr[i] == -1.0)
443     {
444         distractionArray(Dom_data, adres.second);
445         continue;
446     }
447
448     for (int j = 0; j < adres.second; j++) {
449         Result_data[j][0] = count_dom_in_row(Dom_data[j], count);
450         arr[j] = Result_data[j][0];
451         for (int k = 0; k < kmax_result_size; k++) {
452             kMax[j][k] = 0;
453         }
454     }

```

```

455
456
457 for (int j = 0; j < adres.second; j++) {
458     int temp_dom = count_max_pow(arr, adres.second,
459         ↪ Result_data[j][0]);
460     int temp_block = block_in_row(Dom_data[j], count);
461     kMax[j][0] = count_clear_dom(Dom_data[j], count) + 0 +
462         ↪ count_ecv_dom(Dom_data[j], count);
463     kMax[j][1] = count_clear_dom(Dom_data[j], count) + 0;
464     kMax[j][2] = count_clear_dom(Dom_data[j], count) +
465         ↪ count_ecv_dom(Dom_data[j], count);
466     kMax[j][3] = count_clear_dom(Dom_data[j], count);
467
468     Result_data[j][7] += count_dom_pow_in_row(Dom_data[j],
469         ↪ count);
470
471     Result_data[j][8] += count_dom_pow_in_row(Dom_data[j], count)
472         ↪ * Pow_arr[i];
473
474     if (kMax[j][3] == (adres.second - 1))
475         Mull_koef = 2;
476     else Mull_koef = 0;
477     Result_data[j][10] += count_dom_pow_in_row(kMax[j],
478         ↪ kmax_result_size) * Pow_arr[i];
479     Result_data[j][12] += count_dom_pow_in_row(kMax[j],
480         ↪ kmax_result_size) * Pow_arr[i] * Mull_koef;
481
482     if (temp_dom == 1) {
483         Result_data[j][1] += temp_dom;
484         Result_data[j][2] += Pow_arr[i];
485     }
486     if (temp_block == 1) {
487         Result_data[j][4] += block_in_row(Dom_data[j], count);
488         Result_data[j][5] += Pow_arr[i];
489     }
490 }
491 outputmf(Dom_data, adres.second, count, i, f);
492 distractionArray(Dom_data, adres.second);
493 distractionArray(kMax, adres.second);
494 }
495
496 auto* pos_dom = new double[adres.second];
497 auto* pos_block = new double[adres.second];
498 auto* pos_dom_pow = new double[adres.second];
499 auto* pos_kMax_Sjp = new double[adres.second];
500 auto* pos_kMax_Sjm = new double[adres.second];
501
502 for (int j = 0; j < adres.second; j++) {
503     pos_dom[j] = Result_data[j][2];
504     pos_block[j] = Result_data[j][5];
505     pos_dom_pow[j] = Result_data[j][8];
506     pos_kMax_Sjp[j] = Result_data[j][10];
507     pos_kMax_Sjm[j] = Result_data[j][12];
508 }
509
510 for (int j = 0; j < adres.second; j++) {

```

```

506     Result_data[j][3] = count_max_pow(pos_dom, adres.second,
    ↪ Result_data[j][2]);
507     Result_data[j][6] = count_max_pow(pos_block, adres.second,
    ↪ Result_data[j][5]);
508     Result_data[j][9] = count_max_pow(pos_dom_pow, adres.second,
    ↪ Result_data[j][8]);
509     Result_data[j][11] = count_max_pow(pos_kMax_Sjp, adres.second,
    ↪ Result_data[j][10]);
510     Result_data[j][13] = count_max_pow(pos_kMax_Sjm, adres.second,
    ↪ Result_data[j][12]);
511 }
512
513
514 double** Final_data = createArr(adres.second, 6);
515
516 for (int j = 0; j < adres.second; j++) {
517     pos_dom[j] = Result_data[j][3];
518     pos_block[j] = Result_data[j][6];
519     pos_dom_pow[j] = Result_data[j][9];
520     pos_kMax_Sjp[j] = Result_data[j][11];
521     pos_kMax_Sjm[j] = Result_data[j][13];
522 }
523
524
525
526 for (int j = 0; j < adres.second; j++) {
527     Final_data[j][0] = count_point(Result_data[j][3]);
528     Final_data[j][1] = count_point(Result_data[j][6]);
529     Final_data[j][2] = count_point(Result_data[j][9]);
530     Final_data[j][3] = count_point(Result_data[j][11]);
531     Final_data[j][4] = count_point(Result_data[j][13]);
532     Final_data[j][5] = count_point(Result_data[j][3]) +
    ↪ count_point(Result_data[j][6]) +
    ↪ count_point(Result_data[j][9]) +
    ↪ count_point(Result_data[j][11]) +
    ↪ count_point(Result_data[j][13]);
533 }
534
535
536 double* pos_final = new double[adres.second];
537
538 for (int j = 0; j < adres.second; j++) {
539     pos_final[j] = Final_data[j][5];
540 }
541 for (int j = 0; j < adres.second; j++) {
542     Final_data[j][6] = count_max_pow(pos_final, adres.second,
    ↪ Final_data[j][5]);
543 }
544
545
546 setlocale(LC_ALL, "Russian");
547 std::cout << "-----Входящие данные-----" << std::endl;
548 writeArr(0, 0, Dics_data, adres.first, adres.second);
549 std::cout << "-----Механизм доминирования-----" << std::endl;
550 writeArr(0, 1, Result_data, adres.second, 4);
551 std::cout << "-----Механизм блокировки-----" << std::endl;
552 writeArr(0, 4, Result_data, adres.second, 7);
553 std::cout << "-----Турнирный механизм-----" << std::endl;
554 writeArr(0, 8, Result_data, adres.second, 10);

```

```

555     std::cout << "-----Механизм k-Мак вариантов-----" << std::endl;
556     writeArr(0, 10, Result_data, adres.second, 14);
557     std::cout << "-----Механизмы (бальная система)-----" << std::endl;
558     writeArrTheBest(Final_data, adres.second, 7.0, 1);
559
560
561     distractionArray(Result_data, adres.second);
562     distractionArray(Dics_data, adres.first);
563
564     f.close();
565 }

```