Оглавление

[Оглавление 1](#_Toc74187643)

[Постановка задачи 2](#_Toc74187644)

[Многокритериальная оптимизация 3](#_Toc74187645)

[Парето 3](#_Toc74187646)

[Практический расчет задачи 3](#_Toc74187647)

[Программная реализация 3](#_Toc74187648)

[Выводы 9](#_Toc74187649)

[Сужение Парето 10](#_Toc74187650)

[Практический расчет задачи 10](#_Toc74187651)

[Программная реализация 11](#_Toc74187652)

[Выводы 17](#_Toc74187653)

[Метод Электра 18](#_Toc74187654)

[Практический расчет задачи 18](#_Toc74187655)

[Программная реализация 26](#_Toc74187656)

[Выводы 27](#_Toc74187657)

[МАИ 28](#_Toc74187658)

[Практический расчет задачи 28](#_Toc74187659)

[Программная реализация 33](#_Toc74187660)

[Выводы 35](#_Toc74187661)

[Линейное программирование 36](#_Toc74187662)

[Графический метод 36](#_Toc74187663)

[Выводы 36](#_Toc74187664)

[Симплекс метод 37](#_Toc74187665)

[Практический расчет задачи 37](#_Toc74187666)

[Программная реализация 45](#_Toc74187667)

[Выводы 48](#_Toc74187668)

[Двойственная задача 49](#_Toc74187669)

[Практический расчет задачи 49](#_Toc74187670)

[Программная реализация 51](#_Toc74187671)

[Выводы 55](#_Toc74187672)

[Транспортная задача 56](#_Toc74187673)

[Практический расчет задачи 56](#_Toc74187674)

[Программная реализация 64](#_Toc74187675)

[Выводы 68](#_Toc74187676)

[Заключение 69](#_Toc74187677)

[Литература 70](#_Toc74187678)

Постановка задачи

Раздел 1:

Использовать сужение Парето, метод Электра и МАИ для альтернатив:

Меркурий

Венера

Земля

Марс

Юпитер

Сатурн

Уран

Нептун

Плутон

Критерии:

Температура (-)

Плотность (+)

Вода (+)

Вулканы (-)

Атмосфера (+)

Раздел 2:

Решить задачу: Предприятие может работать по двум технологическим процессам, причем за единицу времени по 1 технологии выпускает 260 изделий, по 2 – 300 изделий. В таблице указаны затраты каждого ресурса в единицу времени.

|  |  |  |  |
| --- | --- | --- | --- |
| Ресурсы | 1 | 2 | Объем ресурса |
| Сырье | 16 | 12 | 1200 |
| Электроэнергия | 0.2 | 0.4 | 30 |
| Накладные расходы | 6 | 5 | 600 |
| Зарплата | 3 | 4 | 300 |

Язык программирования: Java

Многокритериальная оптимизация

Многокритериальная оптимизация, или программирование— это процесс одновременной оптимизации двух или более конфликтующих целевых функций в заданной области определения.

Парето

Практический расчет задачи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | Меркурий | 427 | 5.43 | 0 | 423 | 0 |
| 2 | Венера | 600 | 5.42 | 0 | 1023 | 90 |
| 3 | Земля | 20 | 5.5 | 5000 | 56 | 1 |
| 4 | Марс | -40 | 5.4 | 5 | 100 | 2 |
| 5 | Юпитер | -108 | 0.6 | 300 | 0 | 3000 |
| 6 | Сатурн | -109 | 0.7 | 300 | 0 | 3000 |
| 7 | Уран | -100 | 0.8 | 100 | 0 | 2000 |
| 8 | Нептун | -150 | 0.8 | 200 | 0 | 2500 |
| 9 | Плутон | -150 | 5.1 | 0 | 40 | 0.5 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | X | X | X | X | X | X | X | X | X |
| 2 | H | X | X | X | X | X | X | X | X |
| 3 | A3 | H | X | X | X | X | X | X | X |
| 4 | H | H | H | X | X | X | X | X | X |
| 5 | H | H | H | H | X | X | X | X | X |
| 6 | H | H | H | H | A6 | X | X | X | X |
| 7 | H | H | H | H | H | H | X | X | X |
| 8 | H | H | H | H | H | H | A8 | X | X |
| 9 | H | H | H | H | H | H | H | H | X |

Программная реализация

Solution.java

import First.Alternative;  
import First.Pareto;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Solution {  
 public static void main(String[] args) {  
 List<Alternative> alternatives = new ArrayList<>();  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 names.add("Температура");  
 names.add("Плотность");  
 names.add("Вода");  
 names.add("Вулканы");  
 names.add("Атмосфера");  
 importance.add("Температура");  
 importance.add("Атмосфера");  
 importance.add("Вода");  
 importance.add("Вулканы");  
 importance.add("Плотность");  
 List<Alternative.Criteria> narrowing = new ArrayList<>();  
 narrowing.add(new Alternative.Criteria("Температура", 100, true));  
 narrowing.add(new Alternative.Criteria("Вода", 3, false));  
 narrowing.add(new Alternative.Criteria("Атмосфера", 0, false));  
 heights.add(false);  
 heights.add(true);  
 heights.add(true);  
 heights.add(false);  
 heights.add(true);  
 List<Double> counts = new ArrayList<>();  
 counts.add(427.);  
 counts.add(5.43);  
 counts.add(0.);  
 counts.add(423.);  
 counts.add(0.);  
 alternatives.add(new Alternative("Меркурий", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(600.);  
 counts.add(5.42);  
 counts.add(0.);  
 counts.add(1023.);  
 counts.add(90.);  
 alternatives.add(new Alternative("Венера", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(20.);  
 counts.add(5.5);  
 counts.add(5000.);  
 counts.add(56.);  
 counts.add(1.);  
 alternatives.add(new Alternative("Земля", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-40.);  
 counts.add(5.4);  
 counts.add(5.);  
 counts.add(100.);  
 counts.add(2.);  
 alternatives.add(new Alternative("Марс", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-108.);  
 counts.add(0.6);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Юпитер", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-109.);  
 counts.add(0.7);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Сатурн", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-100.);  
 counts.add(0.8);  
 counts.add(100.);  
 counts.add(0.);  
 counts.add(2000.);  
 alternatives.add(new Alternative("Уран", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(0.8);  
 counts.add(200.);  
 counts.add(0.);  
 counts.add(2500.);  
 alternatives.add(new Alternative("Нептун", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(5.1);  
 counts.add(0.);  
 counts.add(40.);  
 counts.add(0.5);  
 alternatives.add(new Alternative("Плутон", names, heights, counts));  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*narrowing*(alternatives, narrowing);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*paretoMethod*(alternatives);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*importance*(alternatives, importance);  
 System.*out*.println();  
 System.*out*.println(alternatives);  
 }  
}

Alternative.java

package First;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
  
public class Alternative implements Comparable<Alternative> {  
 private final String name;  
 private final List<Criteria> criterias = new ArrayList<>();  
  
 public String getName() {  
 return name;  
 }  
  
 public Alternative(String name, List<String> nCriterias, List<Boolean> bCriterias, List<Double> iCriterias) {  
 this.name = name;  
 for (int i = 0; i < nCriterias.size(); i++)  
 criterias.add(new Criteria(nCriterias.get(i), iCriterias.get(i), bCriterias.get(i)));  
 }  
  
 public boolean checkToNarrowing(List<Criteria> criterias2) {  
 for (Criteria criteria : criterias2) {  
 if (criteria.height && criterias.get(criterias.indexOf(criteria)).count > criteria.count)  
 return true;  
 if (!criteria.height && criterias.get(criterias.indexOf(criteria)).count < criteria.count)  
 return true;  
 }  
 return false;  
 }  
  
 public double getCount(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.count;  
 return 0;  
 }  
  
 public boolean getHeight(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.height;  
 return false;  
 }  
  
 public int compareTo(Alternative o) {  
 int res = 0;  
 for (int i = 0; i < criterias.size(); i++) {  
 if (criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count) {  
 if (res < 0)  
 return 0;  
 res++;  
 } else if (criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count)  
 if (res > 0)  
 return 0;  
 res--;  
 }  
 return res;  
 }  
  
 public String toString() {  
 StringBuilder builder = new StringBuilder("Вариант ").append(name).append(":");  
 for (Criteria criteria : criterias)  
 builder.append(", ").append(criteria.criteriaName).append(" = ").append(criteria.count);  
 return builder.toString();  
 }  
  
  
 public static class Criteria {  
 private final String criteriaName;  
 private final double count;  
 private final boolean height;  
  
  
 public Criteria(String criteriaName, double count, boolean height) {  
 this.criteriaName = criteriaName;  
 this.count = count;  
 this.height = height;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Criteria criteria = (Criteria) o;  
 return Objects.*equals*(criteriaName, criteria.criteriaName);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(criteriaName);  
 }  
  
 }  
}

Pareto.java

package First;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Pareto {  
 public static class ParetoHash {  
 public List<Alternative.Criteria> criteriaToNarrowing;  
 public List<String> importance;  
 public List<Alternative> alternatives;  
  
 public ParetoHash(List<Alternative.Criteria> criteriaToNarrowing, List<String> importance, List<Alternative> alternatives) {  
 this.criteriaToNarrowing = criteriaToNarrowing;  
 this.importance = importance;  
 this.alternatives = alternatives;  
 }  
 }  
  
 public static ParetoHash in() throws IOException {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
 List<Alternative> alternatives = new ArrayList<>();  
 System.*out*.println("Введите количество критериев");  
 int n = Integer.*parseInt*(reader.readLine());  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<Alternative.Criteria> criteriaToNarrowing = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите имя для критерия");  
 names.add(reader.readLine());  
 System.*out*.println("Критерий положительный или отрицательный(true/false)?");  
 heights.add(Boolean.*parseBoolean*(reader.readLine()));  
 System.*out*.println("Хотите задать верхнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение верхней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), true));  
 }  
 System.*out*.println("Хотите задать нижнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение нижней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), false));  
 }  
 }  
 System.*out*.println("Напишите название критерия для добавления его в приоритетность или exit для продолжения");  
 String impStr = reader.readLine();  
 while (!impStr.equals("exit")) {  
 importance.add(impStr);  
 impStr = reader.readLine();  
 }  
 System.*out*.println("Введите количество альтернатив");  
 n = Integer.*parseInt*(reader.readLine());  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите название варианта");  
 String name = reader.readLine();  
 List<Double> counts = new ArrayList<>();  
 for (String na : names) {  
 System.*out*.println("Введите значение для критерия " + na);  
 counts.add(Double.*parseDouble*(reader.readLine()));  
 }  
 alternatives.add(new Alternative(name, names, heights, counts));  
 }  
 return new ParetoHash(criteriaToNarrowing, importance, alternatives);  
 }  
  
 public static void paretoMethod(List<Alternative> alternatives) {  
 label:  
 for (int i = 0; i < alternatives.size(); ) {  
 for (int j = i + 1; j < alternatives.size(); ) {  
 int res = alternatives.get(i).compareTo(alternatives.get(j));  
 if (res < 0) {  
 System.*out*.println(alternatives.get(j).getName() + " доминирует над " + alternatives.get(i).getName());  
 alternatives.remove(i);  
 continue label;  
 } else if (res > 0) {  
 System.*out*.println(alternatives.get(i).getName() + " доминирует над " + alternatives.get(j).getName());  
 alternatives.remove(j);  
 continue;  
 }  
 j++;  
 }  
 i++;  
 }  
 }  
  
 public static void narrowing(List<Alternative> alternatives, List<Alternative.Criteria> criteries) {  
 for (int i = 0; i < alternatives.size(); ) {  
 if (alternatives.get(i).checkToNarrowing(criteries)) {  
 System.*out*.println("Вариант " + alternatives.get(i).getName() + " исключается в результате сужения");  
 alternatives.remove(alternatives.get(i));  
 } else  
 i++;  
 }  
 }  
  
 public static void importance(List<Alternative> alternatives, List<String> criteries) {  
 for (String criteria : criteries) {  
 if (alternatives.size() == 1)  
 return;  
 boolean height = alternatives.get(0).getHeight(criteria);  
 List<Alternative> best = new ArrayList<>();  
 for (int i = 0; i < alternatives.size(); ) {  
 Alternative alternative = alternatives.get(i);  
 if (best.isEmpty() || (height && alternative.getCount(criteria) > best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) < best.get(0).getCount(criteria))) {  
 alternatives.removeAll(best);  
 if (!best.isEmpty()) {  
 StringBuilder builder = new StringBuilder("Варианты ");  
 for (Alternative alternative1 : best) {  
 builder.append(alternative1.getName()).append(", ");  
 alternatives.remove(alternative1);  
 }  
 builder.append(" проигрывают варианту ").append(alternative.getName()).append(" по критерию ").append(criteria);  
 System.*out*.println(builder);  
 }  
 best = new ArrayList<>();  
 best.add(alternative);  
 i = 1;  
 } else if ((height && alternative.getCount(criteria) < best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) > best.get(0).getCount(criteria))) {  
 System.*out*.println("Вариант " + alternative.getName() + " проигрывает варианту " + best.get(0).getName());  
 alternatives.remove(alternative);  
 }  
 else {  
 best.add(alternative);  
 i++;  
 }  
 }  
 }  
 }  
  
 public static void out(List<Alternative> alternatives) {  
 for (Alternative alternative : alternatives)  
 System.*out*.println(alternative);  
 }  
}

Выводы

Метод Парето очень эффективен для решения многокритериальных задач

Сужение Парето

Практический расчет задачи

Сужение можно осуществлять с помощью указания нижних и верхних границ критериев. Если в каком-то варианте какой-то критерий в указанные границы не попадает, то данные вариант из множества Парето исключается. Укажем, например, для предыдущей практической верхнюю границу для температуры в 100 градусов и нижнюю для воды в 3 условных единицы и для атмосферы в 0 условных единиц. Получаем:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | ~~Меркурий~~ | 427 | 5.43 | 0 | 423 | 0 |
| 2 | ~~Венера~~ | 600 | 5.42 | 0 | 1023 | 90 |
| 3 | Земля | 20 | 5.5 | 5000 | 56 | 1 |
| 4 | Марс | -40 | 5.4 | 5 | 100 | 2 |
| 5 | Юпитер | -108 | 0.6 | 300 | 0 | 3000 |
| 6 | Сатурн | -109 | 0.7 | 300 | 0 | 3000 |
| 7 | Уран | -100 | 0.8 | 100 | 0 | 2000 |
| 8 | Нептун | -150 | 0.8 | 200 | 0 | 2500 |
| 9 | ~~Плутон~~ | -150 | 5.1 | 0 | 40 | 0.5 |

Меркурий на проходит по температуре, атмосфере и воде, Венера не проходит по температуре и воде, Плутон не проходит по воде.

Также мы можем использовать выделение приоритетности некоторых критериев над другими. Допустим, если мы скажем, что из оставшегося множества нас больше всего волнует критерий “Вода”, то мы получим:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | Земля | 20 | 5.5 | 5000 | 56 | 1 |
| 2 | ~~Марс~~ | -40 | 5.4 | 5 | 100 | 2 |
| 3 | ~~Юпитер~~ | -108 | 0.6 | 300 | 0 | 3000 |
| 4 | ~~Сатурн~~ | -109 | 0.7 | 300 | 0 | 3000 |
| 5 | ~~Уран~~ | -100 | 0.8 | 100 | 0 | 2000 |
| 6 | ~~Нептун~~ | -150 | 0.8 | 200 | 0 | 2500 |

А если же нас больше всего интересует критерий “Температура”, то выходит:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | ~~Земля~~ | 20 | 5.5 | 5000 | 56 | 1 |
| 2 | ~~Марс~~ | -40 | 5.4 | 5 | 100 | 2 |
| 3 | ~~Юпитер~~ | -108 | 0.6 | 300 | 0 | 3000 |
| 4 | ~~Сатурн~~ | -109 | 0.7 | 300 | 0 | 3000 |
| 5 | ~~Уран~~ | -100 | 0.8 | 100 | 0 | 2000 |
| 6 | Нептун | -150 | 0.8 | 200 | 0 | 2500 |

Программная реализация

import First.Alternative;  
import First.Pareto;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Solution {  
 public static void main(String[] args) {  
 List<Alternative> alternatives = new ArrayList<>();  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 names.add("Температура");  
 names.add("Плотность");  
 names.add("Вода");  
 names.add("Вулканы");  
 names.add("Атмосфера");  
 importance.add("Температура");  
 importance.add("Атмосфера");  
 importance.add("Вода");  
 importance.add("Вулканы");  
 importance.add("Плотность");  
 List<Alternative.Criteria> narrowing = new ArrayList<>();  
 narrowing.add(new Alternative.Criteria("Температура", 100, true));  
 narrowing.add(new Alternative.Criteria("Вода", 3, false));  
 narrowing.add(new Alternative.Criteria("Атмосфера", 0, false));  
 heights.add(false);  
 heights.add(true);  
 heights.add(true);  
 heights.add(false);  
 heights.add(true);  
 List<Double> counts = new ArrayList<>();  
 counts.add(427.);  
 counts.add(5.43);  
 counts.add(0.);  
 counts.add(423.);  
 counts.add(0.);  
 alternatives.add(new Alternative("Меркурий", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(600.);  
 counts.add(5.42);  
 counts.add(0.);  
 counts.add(1023.);  
 counts.add(90.);  
 alternatives.add(new Alternative("Венера", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(20.);  
 counts.add(5.5);  
 counts.add(5000.);  
 counts.add(56.);  
 counts.add(1.);  
 alternatives.add(new Alternative("Земля", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-40.);  
 counts.add(5.4);  
 counts.add(5.);  
 counts.add(100.);  
 counts.add(2.);  
 alternatives.add(new Alternative("Марс", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-108.);  
 counts.add(0.6);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Юпитер", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-109.);  
 counts.add(0.7);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Сатурн", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-100.);  
 counts.add(0.8);  
 counts.add(100.);  
 counts.add(0.);  
 counts.add(2000.);  
 alternatives.add(new Alternative("Уран", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(0.8);  
 counts.add(200.);  
 counts.add(0.);  
 counts.add(2500.);  
 alternatives.add(new Alternative("Нептун", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(5.1);  
 counts.add(0.);  
 counts.add(40.);  
 counts.add(0.5);  
 alternatives.add(new Alternative("Плутон", names, heights, counts));  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*narrowing*(alternatives, narrowing);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*paretoMethod*(alternatives);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*importance*(alternatives, importance);  
 System.*out*.println();  
 System.*out*.println(alternatives);  
 }  
}

**Alternative.java**

package First;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
  
public class Alternative implements Comparable<Alternative> {  
 private final String name;  
 private final List<Criteria> criterias = new ArrayList<>();  
  
 public String getName() {  
 return name;  
 }  
  
 public Alternative(String name, List<String> nCriterias, List<Boolean> bCriterias, List<Double> iCriterias) {  
 this.name = name;  
 for (int i = 0; i < nCriterias.size(); i++)  
 criterias.add(new Criteria(nCriterias.get(i), iCriterias.get(i), bCriterias.get(i)));  
 }  
  
 public boolean checkToNarrowing(List<Criteria> criterias2) {  
 for (Criteria criteria : criterias2) {  
 if (criteria.height && criterias.get(criterias.indexOf(criteria)).count > criteria.count)  
 return true;  
 if (!criteria.height && criterias.get(criterias.indexOf(criteria)).count < criteria.count)  
 return true;  
 }  
 return false;  
 }  
  
 public double getCount(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.count;  
 return 0;  
 }  
  
 public boolean getHeight(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.height;  
 return false;  
 }  
  
 public int compareTo(Alternative o) {  
 int res = 0;  
 for (int i = 0; i < criterias.size(); i++) {  
 if (criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count) {  
 if (res < 0)  
 return 0;  
 res++;  
 } else if (criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count)  
 if (res > 0)  
 return 0;  
 res--;  
 }  
 return res;  
 }  
  
 public String toString() {  
 StringBuilder builder = new StringBuilder("Вариант ").append(name).append(":");  
 for (Criteria criteria : criterias)  
 builder.append(", ").append(criteria.criteriaName).append(" = ").append(criteria.count);  
 return builder.toString();  
 }  
  
  
 public static class Criteria {  
 private final String criteriaName;  
 private final double count;  
 private final boolean height;  
  
  
 public Criteria(String criteriaName, double count, boolean height) {  
 this.criteriaName = criteriaName;  
 this.count = count;  
 this.height = height;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Criteria criteria = (Criteria) o;  
 return Objects.*equals*(criteriaName, criteria.criteriaName);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(criteriaName);  
 }  
  
 }  
}

**Pareto.java**

package First;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Pareto {  
 public static class ParetoHash {  
 public List<Alternative.Criteria> criteriaToNarrowing;  
 public List<String> importance;  
 public List<Alternative> alternatives;  
  
 public ParetoHash(List<Alternative.Criteria> criteriaToNarrowing, List<String> importance, List<Alternative> alternatives) {  
 this.criteriaToNarrowing = criteriaToNarrowing;  
 this.importance = importance;  
 this.alternatives = alternatives;  
 }  
 }  
  
 public static ParetoHash in() throws IOException {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
 List<Alternative> alternatives = new ArrayList<>();  
 System.*out*.println("Введите количество критериев");  
 int n = Integer.*parseInt*(reader.readLine());  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<Alternative.Criteria> criteriaToNarrowing = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите имя для критерия");  
 names.add(reader.readLine());  
 System.*out*.println("Критерий положительный или отрицательный(true/false)?");  
 heights.add(Boolean.*parseBoolean*(reader.readLine()));  
 System.*out*.println("Хотите задать верхнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение верхней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), true));  
 }  
 System.*out*.println("Хотите задать нижнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение нижней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), false));  
 }  
 }  
 System.*out*.println("Напишите название критерия для добавления его в приоритетность или exit для продолжения");  
 String impStr = reader.readLine();  
 while (!impStr.equals("exit")) {  
 importance.add(impStr);  
 impStr = reader.readLine();  
 }  
 System.*out*.println("Введите количество альтернатив");  
 n = Integer.*parseInt*(reader.readLine());  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите название варианта");  
 String name = reader.readLine();  
 List<Double> counts = new ArrayList<>();  
 for (String na : names) {  
 System.*out*.println("Введите значение для критерия " + na);  
 counts.add(Double.*parseDouble*(reader.readLine()));  
 }  
 alternatives.add(new Alternative(name, names, heights, counts));  
 }  
 return new ParetoHash(criteriaToNarrowing, importance, alternatives);  
 }  
  
 public static void paretoMethod(List<Alternative> alternatives) {  
 label:  
 for (int i = 0; i < alternatives.size(); ) {  
 for (int j = i + 1; j < alternatives.size(); ) {  
 int res = alternatives.get(i).compareTo(alternatives.get(j));  
 if (res < 0) {  
 System.*out*.println(alternatives.get(j).getName() + " доминирует над " + alternatives.get(i).getName());  
 alternatives.remove(i);  
 continue label;  
 } else if (res > 0) {  
 System.*out*.println(alternatives.get(i).getName() + " доминирует над " + alternatives.get(j).getName());  
 alternatives.remove(j);  
 continue;  
 }  
 j++;  
 }  
 i++;  
 }  
 }  
  
 public static void narrowing(List<Alternative> alternatives, List<Alternative.Criteria> criteries) {  
 for (int i = 0; i < alternatives.size(); ) {  
 if (alternatives.get(i).checkToNarrowing(criteries)) {  
 System.*out*.println("Вариант " + alternatives.get(i).getName() + " исключается в результате сужения");  
 alternatives.remove(alternatives.get(i));  
 } else  
 i++;  
 }  
 }  
  
 public static void importance(List<Alternative> alternatives, List<String> criteries) {  
 for (String criteria : criteries) {  
 if (alternatives.size() == 1)  
 return;  
 boolean height = alternatives.get(0).getHeight(criteria);  
 List<Alternative> best = new ArrayList<>();  
 for (int i = 0; i < alternatives.size(); ) {  
 Alternative alternative = alternatives.get(i);  
 if (best.isEmpty() || (height && alternative.getCount(criteria) > best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) < best.get(0).getCount(criteria))) {  
 alternatives.removeAll(best);  
 if (!best.isEmpty()) {  
 StringBuilder builder = new StringBuilder("Варианты ");  
 for (Alternative alternative1 : best) {  
 builder.append(alternative1.getName()).append(", ");  
 alternatives.remove(alternative1);  
 }  
 builder.append(" проигрывают варианту ").append(alternative.getName()).append(" по критерию ").append(criteria);  
 System.*out*.println(builder);  
 }  
 best = new ArrayList<>();  
 best.add(alternative);  
 i = 1;  
 } else if ((height && alternative.getCount(criteria) < best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) > best.get(0).getCount(criteria))) {  
 System.*out*.println("Вариант " + alternative.getName() + " проигрывает варианту " + best.get(0).getName());  
 alternatives.remove(alternative);  
 }  
 else {  
 best.add(alternative);  
 i++;  
 }  
 }  
 }  
 }  
  
 public static void out(List<Alternative> alternatives) {  
 for (Alternative alternative : alternatives)  
 System.*out*.println(alternative);  
 }  
}

Выводы

Сужение Парето очень эффективно для решения многокритериальных задач

Метод Электра

Практический расчет задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерии | Вес критериев | Шкала | Код | Стремление |
| Температура | 5 | Высокая  Средняя  Низкая | 15  10  5 | Min |
| Атмосфера | 5 | Плотная  Разряженная  Отсутствует | 15  10  5 | Max |
| Вода | 4 | Много  Мало  Отсутствует | 15  10  5 | Max |
| Вулканы | 2 | Много  Мало  Отсутствуют | 15  10  5 | Min |
| Плотность | 1 | Высокая  Газовый гигант | 2  1 | max |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | Меркурий | 15 | 2 | 5 | 15 | 5 |
| 2 | Венера | 15 | 2 | 10 | 15 | 15 |
| 3 | Земля | 10 | 2 | 15 | 10 | 15 |
| 4 | Марс | 5 | 2 | 10 | 10 | 10 |
| 5 | Юпитер | 5 | 1 | 10 | 5 | 15 |
| 6 | Сатурн | 5 | 1 | 10 | 5 | 15 |
| 7 | Уран | 5 | 1 | 5 | 5 | 15 |
| 8 | Нептун | 5 | 1 | 15 | 5 | 15 |
| 9 | Плутон | 5 | 2 | 10 | 5 | 10 |

Pi – вес альтернативы под I, Pj – вес альтернативы J, Dij – отношение альтернативы I к альтернативе J

Для I = 1, J = 2:

Температура: 15 = 15 (-)

Плотность: 2 = 2 (-)

Вода: 5 < 10 (P2 + 4)

Вулканы: 15 = 15 (-)

Атмосфера: 5 < 15 (P2 + 5)

P2 = 9, P1 = 0 => D21 = БЕСКОНЕЧНОСТЬ

Для I = 1, J = 3:

Температура: 15 > 10 (P3 + 5)

Плотность: 2 = 2 (-)

Вода: 5 < 15 (P3 + 4)

Вулканы: 15 > 10 (P3 + 2)

Атмосфера: 5 < 15 (P3 + 5)

P3 = 16, P1 = 0 => D31 = БЕСКОНЕЧНОСТЬ

Для I = 1, J = 4:

Температура: 15 > 5 (P4 + 5)

Плотность: 2 = 2 (-)

Вода: 5 < 10 (P4 + 4)

Вулканы: 15 > 10 (P4 + 2)

Атмосфера: 5 < 10 (P4 + 5)

P2 = 16, P1 = 0 => D41 = БЕСКОНЕЧНОСТЬ

Для I = 1, J = 5:

Температура: 15 > 5 (P5 + 5)

Плотность: 2 > 1 (P1 + 1)

Вода: 5 < 10 (P5 + 4)

Вулканы: 15 > 5 (P5 + 2)

Атмосфера: 5 < 15 (P5 + 5)

P5 = 16, P1 = 1 => D51 = 16

Для I = 1, J = 6:

Температура: 15 > 5 (P6 + 5)

Плотность: 2 > 1 (P1 + 1)

Вода: 5 < 10 (P6 + 4)

Вулканы: 15 > 5 (P6 + 2)

Атмосфера: 5 < 15 (P6 + 5)

P5 = 16, P1 = 1 => D61 = 16

Для I = 1, J = 7:

Температура: 15 > 5 (P7 + 5)

Плотность: 2 > 1 (P1 + 1)

Вода: 5 = 5 (-)

Вулканы: 15 > 5 (P7 + 2)

Атмосфера: 5 < 15 (P7 + 5)

P7 = 12, P1 = 1 => D71 = 12

Для I = 1, J = 8:

Температура: 15 > 5 (P8 + 5)

Плотность: 2 > 1 (P1 + 1)

Вода: 5 < 15 (P8 + 4)

Вулканы: 15 > 5 (P8 + 2)

Атмосфера: 5 < 15 (P8 + 5)

P8 = 16, P1 = 1 => D81 = 16

Для I = 1, J = 9:

Температура: 15 > 5 (P9 + 5)

Плотность: 2 = 2 (-)

Вода: 5 < 10 (P9 + 4)

Вулканы: 15 > 5 (P9 + 2)

Атмосфера: 5 < 10 (P9 + 5)

P9 = 16, P1 = 0 => D91 = БЕСКОНЕЧНОСТЬ

Для I = 2, J = 3:

Температура: 15 > 10 (P3 + 5)

Плотность: 2 = 2 (-)

Вода: 10 < 15 (P3 + 4)

Вулканы: 15 > 10 (P3 + 2)

Атмосфера: 15 = 15 (-)

P3 = 11, P2 = 0 => D32 = БЕСКОНЕЧНОСТЬ

Для I = 2, J = 4:

Температура: 15 > 5 (P4 + 5)

Плотность: 2 = 2 (-)

Вода: 10 = 10 (-)

Вулканы: 15 > 10 (P4 + 2)

Атмосфера: 15 > 10 (P2 + 5)

P2 = 5, P4 = 7 => D42 = 7/5

Для I = 2, J = 5:

Температура: 15 > 5 (P5 + 5)

Плотность: 2 > 1 (P2 + 1)

Вода: 10 = 10 (-)

Вулканы: 15 > 5 (P5 + 2)

Атмосфера: 15 = 15 (-)

P2 = 1, P5 = 7 => D52 = 7

Для I = 2, J = 6:

Температура: 15 > 5 (P6 + 5)

Плотность: 2 > 1 (P2 + 1)

Вода: 10 = 10 (-)

Вулканы: 15 > 5 (P6 + 2)

Атмосфера: 15 = 15 (-)

P2 = 1, P6 = 7 => D62 = 7

Для I = 2, J = 7:

Температура: 15 > 5 (P7 + 5)

Плотность: 2 > 1 (P2 + 1)

Вода: 10 > 5 (P2 + 4)

Вулканы: 15 > 5 (P7 + 2)

Атмосфера: 15 = 15 (-)

P2 = 5, P7 = 7 => D72 = 7/5

Для I = 2, J = 8:

Температура: 15 > 5 (P8 + 5)

Плотность: 2 > 1 (P2 + 1)

Вода: 10 < 15 (P8 + 4)

Вулканы: 15 > 5 (P8 + 2)

Атмосфера: 15 = 15 (-)

P2 = 1, P8 = 11 => D82 = 11

Для I = 2, J = 9:

Температура: 15 > 5 (P9 + 5)

Плотность: 2 = 2 (-)

Вода: 10 = 10 (-)

Вулканы: 15 > 5 (P9 + 2)

Атмосфера: 15 > 10 (P2 + 5)

P2 = 5, P9 = 7 => D92 = 7/5

Для I = 3, J = 4:

Температура: 10 > 5 (P4 + 5)

Плотность: 2 = 2 (-)

Вода: 15 > 10 (P3 + 4)

Вулканы: 10 = 10 (-)

Атмосфера: 15 > 10 (P3 + 5)

P3 = 9, P4 = 5 => D34 = 9/5

Для I = 3, J = 5:

Температура: 10 > 5 (P5 + 5)

Плотность: 2 > 1 (P3 + 1)

Вода: 15 >10 (P3 + 4)

Вулканы: 10 > 5 (P5 + 2)

Атмосфера: 15 = 15 (-)

P3 = 5, P5 = 7 => D53 = 7/5

Для I = 3, J = 6:

Температура: 10 > 5 (P6 + 5)

Плотность: 2 > 1 (P3 + 1)

Вода: 15 >10 (P3 + 4)

Вулканы: 10 > 5 (P6 + 2)

Атмосфера: 15 = 15 (-)

P3 = 5, P6 = 7 => D63 = 7/5

Для I = 3, J = 7:

Температура: 10 > 5 (P7 + 5)

Плотность: 2 > 1 (P3 + 1)

Вода: 15 > 5 (P3 + 4)

Вулканы: 10 > 5 (P7 + 2)

Атмосфера: 15 = 15 (-)

P3 = 5, P7 = 7 => D73 = 7/5

Для I = 3, J = 8:

Температура: 10 > 5 (P8 + 5)

Плотность: 2 > 1 (P3 + 1)

Вода: 15 = 15 (-)

Вулканы: 10 > 5 (P8 + 2)

Атмосфера: 15 = 15 (-)

P3 = 1, P8 = 7 => D83 = 7

Для I = 3, J = 9:

Температура: 10 > 5 (P9 + 5)

Плотность: 2 = 2 (-)

Вода: 15 > 10 (P3 + 4)

Вулканы: 10 > 5 (P9 + 2)

Атмосфера: 15 > 10 (P3 + 5)

P3 = 9, P9 = 7 => D39 = 9/7

Для I = 4, J = 5:

Температура: 5 = 5 (-)

Плотность: 2 > 1 (P4 + 1)

Вода: 10 = 10 (-)

Вулканы: 10 > 5 (P5 + 2)

Атмосфера: 10 < 15 (P5 + 5)

P4 = 1, P5 = 7 => D54 = 7

Для I = 4, J = 6:

Температура: 5 = 5 (-)

Плотность: 2 > 1 (P4 + 1)

Вода: 10 = 10 (-)

Вулканы: 10 > 5 (P6 + 2)

Атмосфера: 10 < 15 (P6 + 5)

P4 = 1, P6 = 7 => D64 = 7

Для I = 4, J = 7:

Температура: 5 = 5 (-)

Плотность: 2 > 1 (P4 + 1)

Вода: 10 > 5 (P4 + 4)

Вулканы: 10 > 5 (P7 + 2)

Атмосфера: 10 < 15 (P7 + 5)

P4 = 5, P7 = 7 => D74 = 7/5

Для I = 4, J = 8:

Температура: 5 = 5 (-)

Плотность: 2 > 1 (P4 + 1)

Вода: 10 < 15 (P8 + 4)

Вулканы: 10 > 5 (P8 + 2)

Атмосфера: 10 < 15 (P8 + 5)

P4 = 1, P8 = 11 => D84 = 11

Для I = 4, J = 9:

Температура: 5 = 5 (-)

Плотность: 2 = 2 (-)

Вода: 10 = 10 (-)

Вулканы: 10 > 5 (P9 + 2)

Атмосфера: 10 = 10 (-)

P4 = 0, P9 = 2 => D94 = БЕСКОНЕЧНОСТЬ

Для I = 5, J = 6:

Температура: 5 = 5 (-)

Плотность: 1 = 1 (-)

Вода: 10 = 10 (-)

Вулканы: 5 = 5 (-)

Атмосфера: 15 = 15 (-)

P5 = 0, P6 = 0 => D56 = 1

Для I = 5, J = 7:

Температура: 5 = 5 (-)

Плотность: 1 = 1 (-)

Вода: 10 > 5 (P5 + 4)

Вулканы: 5 = 5 (-)

Атмосфера: 15 = 15 (-)

P5 = 4, P7 = 0 => D57 = БЕСКОНЕЧНОСТЬ

Для I = 5, J = 8:

Температура: 5 = 5 (-)

Плотность: 1 = 1 (-)

Вода: 10 < 15 (P8 + 4)

Вулканы: 5 = 5 (-)

Атмосфера: 15 = 15 (-)

P5 = 0, P8 = 4 => D85 = БЕСКОНЕЧНОСТЬ

Для I = 5, J = 9:

Температура: 5 = 5 (-)

Плотность: 1 < 2 (P9 + 1)

Вода: 10 = 10 (-)

Вулканы: 5 = 5 (-)

Атмосфера: 15 > 10 (P5 + 5)

P5 = 5, P9 = 1 => D59 = 5

Для I = 6, J = 7:

Температура: 5 = 5 (-)

Плотность: 1 = 1 (-)

Вода: 10 > 5 (P6 + 4)

Вулканы: 5 = 5 (-)

Атмосфера: 15 = 15 (-)

P6 = 4, P7 = 0 => D67 = БЕСКОНЕЧНОСТЬ

Для I = 6, J = 8:

Температура: 5 = 5 (-)

Плотность: 1 = 1 (-)

Вода: 10 < 15 (P8 + 4)

Вулканы: 5 = 5 (-)

Атмосфера: 15 = 15 (-)

P6 = 0, P8 = 4 => D86 = БЕСКОНЕЧНОСТЬ

Для I = 6, J = 9:

Температура: 5 = 5 (-)

Плотность: 1 < 2 (P9 + 1)

Вода: 10 = 10 (-)

Вулканы: 5 = 5 (-)

Атмосфера: 15 > 10 (P6 + 5)

P6 = 5, P9 = 1 => D69 = 5

Для I = 7, J = 8:

Температура: 5 = 5 (-)

Плотность: 1 = 1 (-)

Вода: 5 < 15 (P8 + 4)

Вулканы: 5 = 5 (-)

Атмосфера: 15 = 15 (-)

P7 = 0, P8 = 4 => D87 = БЕСКОНЕЧНОСТЬ

Для I = 7, J = 9:

Температура: 5 = 5 (-)

Плотность: 1 < 2 (P9 + 1)

Вода: 5 < 10 (P9 + 4)

Вулканы: 5 = 5 (-)

Атмосфера: 15 > 10 (P7 + 5)

P7 = 5, P9 = 5 => D79 = 1

Для I = 8, J = 9:

Температура: 5 = 5 (-)

Плотность: 1 < 2 (P9 + 1)

Вода: 15 > 10 (P8 + 4)

Вулканы: 5 = 5 (-)

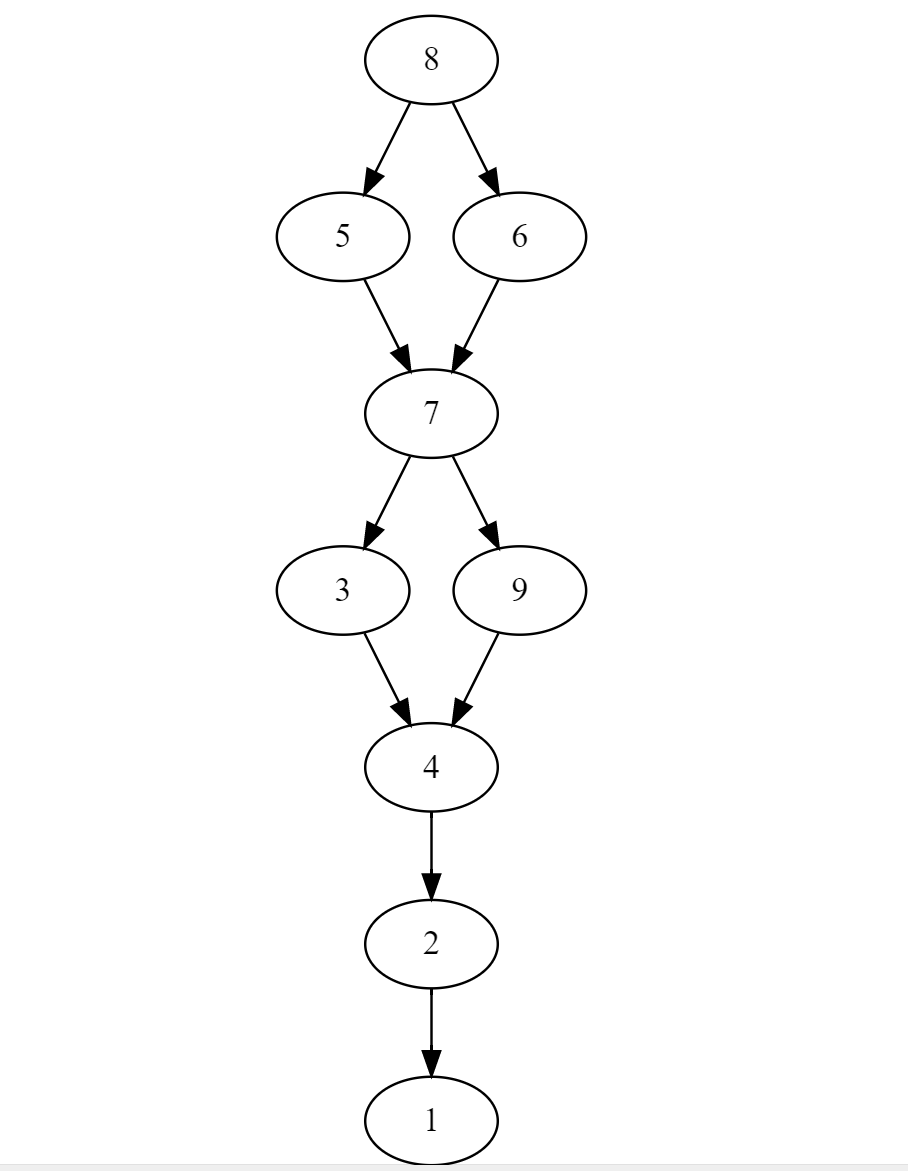
Атмосфера: 15 > 10 (P8 + 5)

P8 = 9, P9 = 1 => D89 = 9

Теперь, на основе полученных значений D, создаем матрицу предпочтений:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | X | - | - | - | - | - | - | - | - |
| 2 | Inf | x | - | - | - | - | - | - | - |
| 3 | Inf | Inf | x | 9/5 | - | - | - | - | 9/7 |
| 4 | Inf | 7/5 | - | x | - | - | - | - | - |
| 5 | 16 | 7 | 7/5 | 7 | x | 1 | inf | - | 5 |
| 6 | 16 | 7 | 7/5 | 7 | 1 | x | inf | - | 5 |
| 7 | 12 | 7/5 | 7/5 | 7/5 | - | - | x | - | 1 |
| 8 | 16 | 11 | 7 | 11 | inf | inf | inf | x | 9 |
| 9 | Inf | 7/5 | - | inf | - | - | 1 | - | x |

Создаем на основании матрицы граф предпочтений:



На основании графа предпочтений мы можем расставить варианты в таком порядке приоритетности (от наилучшего к наихудшему):

Нептун -> Юпитер | Сатурн -> Уран -> Земля | Плутон -> Марс -> Венера -> Меркурий

Программная реализация

package Second;  
  
import First.Alternative;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.Map;  
import java.util.TreeMap;  
  
public class Electro {  
 private final Map<String, Integer> weight;  
  
 private Electro(Map<String, Integer> weight) {  
 this.weight = weight;  
 }  
  
 public double compare(Alternative alternative1, Alternative alternative2) {  
 int first = 0;  
 int second = 0;  
 System.*out*.println();  
 for (Map.Entry<String, Integer> entry : weight.entrySet()) {  
 boolean height = alternative1.getHeight(entry.getKey());  
 if (height && alternative1.getCount(entry.getKey()) > alternative2.getCount(entry.getKey())  
 || !height && alternative1.getCount(entry.getKey()) < alternative2.getCount(entry.getKey())) {  
 first += entry.getValue();  
 if (alternative1.getHeight(entry.getKey()))  
 System.*out*.println("Критерий " + entry.getKey() + " возрастающий критерий");  
 else  
 System.*out*.println("Критерий " + entry.getKey() + " убывающий критерий");  
 System.*out*.println("Альтернатива " + alternative1.getName() + " выигрывает по критерию "  
 + entry.getKey() + " (" + alternative1.getCount(entry.getKey()) + " > " +  
 alternative2.getCount(entry.getKey()) + ") " + " и получает " + entry.getValue() + " баллов");  
 } else if (height && alternative1.getCount(entry.getKey()) < alternative2.getCount(entry.getKey()) || !height && alternative1.getCount(entry.getKey()) > alternative2.getCount(entry.getKey())) {  
 second += entry.getValue();  
 if (alternative1.getHeight(entry.getKey()))  
 System.*out*.println("Критерий " + entry.getKey() + " возрастающий критерий");  
 else  
 System.*out*.println("Критерий " + entry.getKey() + " убывающий критерий");  
 System.*out*.println("Альтернатива " + alternative2.getName() + " выигрывает по критерию "  
 + entry.getKey() + " (" + alternative2.getCount(entry.getKey()) + " > " +  
 alternative1.getCount(entry.getKey()) + ") " + " и получает " + entry.getValue() + " баллов");  
 }  
 }  
 System.*out*.println(alternative1.getName() + " = " + first + ", " + alternative2.getName() + " = " + second);  
 System.*out*.println();  
 if (first == second)  
 return 1;  
 return (double) first / second;  
 }  
  
 public static void calculate(List<Alternative> alternatives, Map<String, Integer> weight) {  
 Electro electro = new Electro(weight);  
 TreeMap<Integer, String> map = new TreeMap<>();  
 double[][] matrix = new double[alternatives.size()][alternatives.size()];  
 for (int i = 0; i < alternatives.size(); i++) {  
 matrix[i][i] = -2;  
 for (int j = i; j < alternatives.size(); j++) {  
 double result = electro.compare(alternatives.get(i), alternatives.get(j));  
 if (result == 1) {  
 matrix[i][j] = 1;  
 matrix[j][i] = 1;  
 }  
 else if (result > 1) {  
 matrix[i][j] = result;  
 matrix[j][i] = -2;  
 } else {  
 matrix[i][j] = -2;  
 matrix[j][i] = 1 / result;  
 }  
 }  
 }  
 for (double[] row : matrix) {  
 System.*out*.println(Arrays.*toString*(row));  
 }  
 for (int i = 0; i < matrix.length; i++) {  
 int count = 0;  
 for (int j = 0; j < matrix[i].length; j++) {  
 if (matrix[i][j] == -1 || matrix[i][j] >= 1)  
 count++;  
 }  
 String name = alternatives.get(i).getName();  
 map.computeIfPresent(count, (a, b) -> b + ", " + name);  
 map.putIfAbsent(count, name);  
 }  
 System.*out*.println();  
 System.*out*.println("Результат: ");  
 for (String value : map.descendingMap().values()) {  
 System.*out*.println(value);  
 }  
 }  
}

Выводы

Метод Электра очень эффективен для решения многокритериальных задач

МАИ

Практический расчет задачи

Сначала мы создаем таблицу, в которой указываем условное значение каждой из альтернатив по каждому из критериев (рисунок 1).



Рисунок 1. Таблица значений критериев

Далее мы начинаем создавать таблицы условных отношений. Сначала мы создаем таблицы, в которой попарно сравниваем все критерии. Вычисление результата сравнения происходит путем нормализации интервала между значениями важности критериев по какому-то числу. Допустим, если мы берем нормализацию по 9, то нам необходимо вычислить интервал между сравниваемыми критериями, разделить его на максимальный интервал среди критериев и умножить полученное число на число нормализации (рисунок 2):



Рисунок 2. Сравнение критериев

Приведу пример вычислений для понимания. Сначала вычислим Vi (относительная ценность) для каждой строки:

V1 = (1 \* 5 \* 5 \* 1 \* 0,2)^(1/5) = 1.3797

V2 = (0,2 \* 1 \* 1 \* 0,2 \* 0,111111)^(1/5) = 0.3385

V3 = (0,2 \* 1 \* 1 \* 0,2 \* 0,111111)^(1/5) = 0.3385

V4 = (1 \* 5 \* 5 \* 1 \* 0,2)^(1/5) = 1.3797

V5 = (5 \* 9 \* 9 \* 5 \* 1)^(1/5) = 4.5844

Вычислим сумму всех V:

Vобщ = V1 + V2 + V3 + V4 + V5 = 8.0208

Вычислим Wi (нормализация) для каждой строки:

W1 = V1 / Vобщ = 1.3797 / 8.0208= 0.1720

W2 = V2 / Vобщ = 0.3385/ 8.0208= 0.0422

W3 = V3 / Vобщ = 0.3385/ 8.0208= 0.0422

W4 = V4 / Vобщ = 1.3797/ 8.0208= 0.1720

W5 = V5 / Vобщ = 4.5844/ 8.0208= 0.5716

Получаем вектор приоритетов:

(0.1720, 0.0422, 0.0422, 0.1720, 0.5716)

Проверим согласованность матрицы. Для этого вычислим Si для каждого столбца:

S1 = 1 + 0.2 + 0.2 + 1 + 5= 7.4

S2 = 5 + 1 + 1 + 5 + 9 = 21

S3 = 5 + 1 + 1 + 5 + 9 = 21

S4 = 1 + 0.2 + 0.2 + 1 + 5= 7.4

S5 = 0.2 + 0.2 + 1 + 0.111111 + 0.111111 = 1.6222

Найдем все Pi (проведем нормализацию):

P1 = S1 \* W1= 1.2728

P2 = S2 \* W2= 0.8862

P3 = S3 \* W3= 0.8862

P4 = S4 \* W4= 1.2728

P5 = S5 \* W5= 0.9272

Находим пропорциональность предпочтений:

& = P1 + P2 + P3 + P4 + P5 = 5.2452

Найдем индекс согласованности:

ИС = (& - 5) / (5 – 1) = 0.0613

Найдем отношение согласованности:

ОС = ИС / СИ = 0.0613 / 1.12 = 0.0547

Так как ОС < 0.1, то мы делаем вывод, что матрица согласована.

Далее мы по каждому критерию производим сравнение для всех альтернатив по схожему методу и в итоге получаем ряд таблиц (рисунок 3):

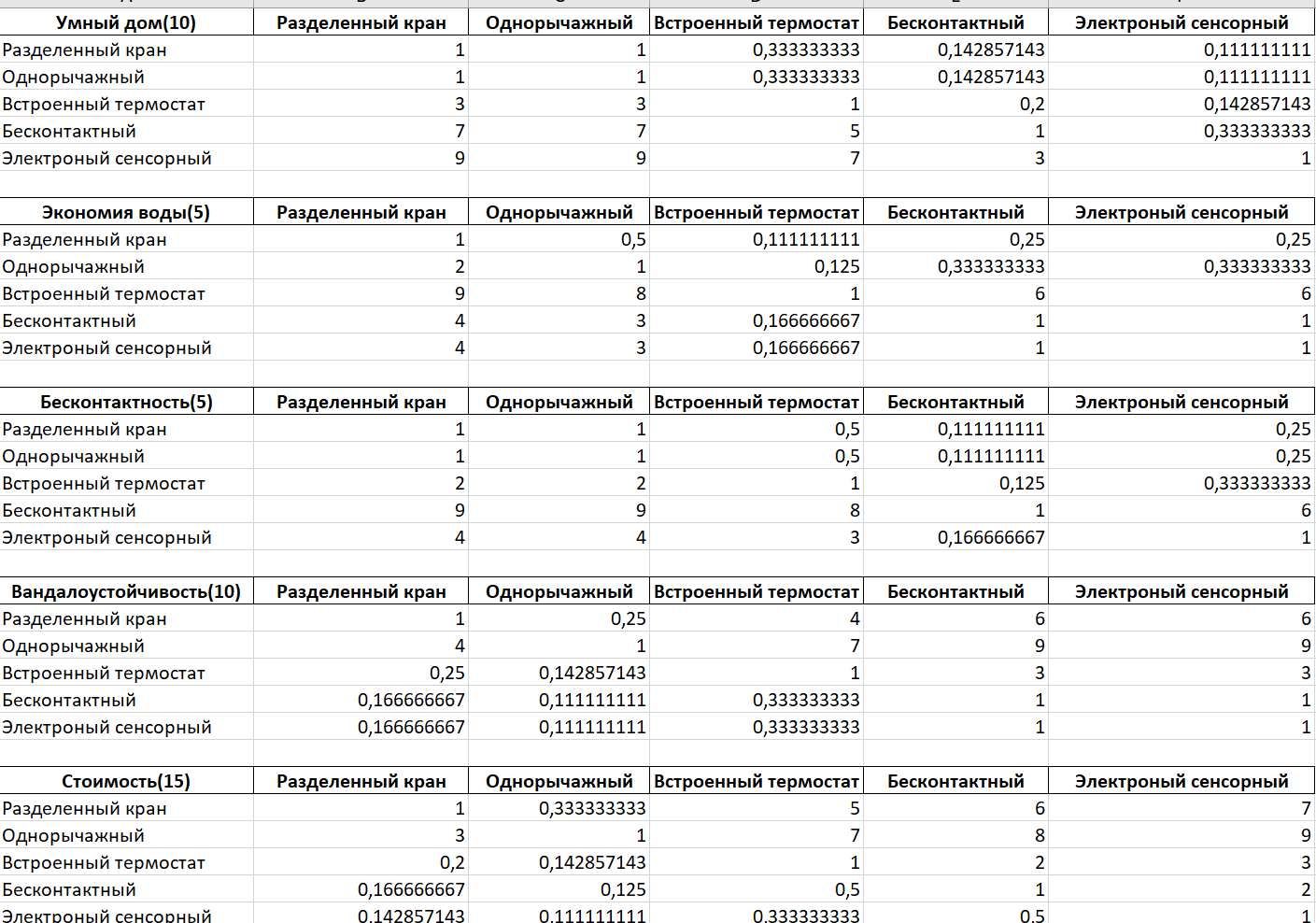


Рисунок 3. Сравнение альтернатив

Далее мы складываем все строчки каждой таблицы и в итоге для каждой таблицы получаем набор чисел, которые мы назовем Vi, где i номер строки в таблице. Разделим каждое Vi на сумму всех Vi для данной таблицы и получим Wi (рисунок 4).

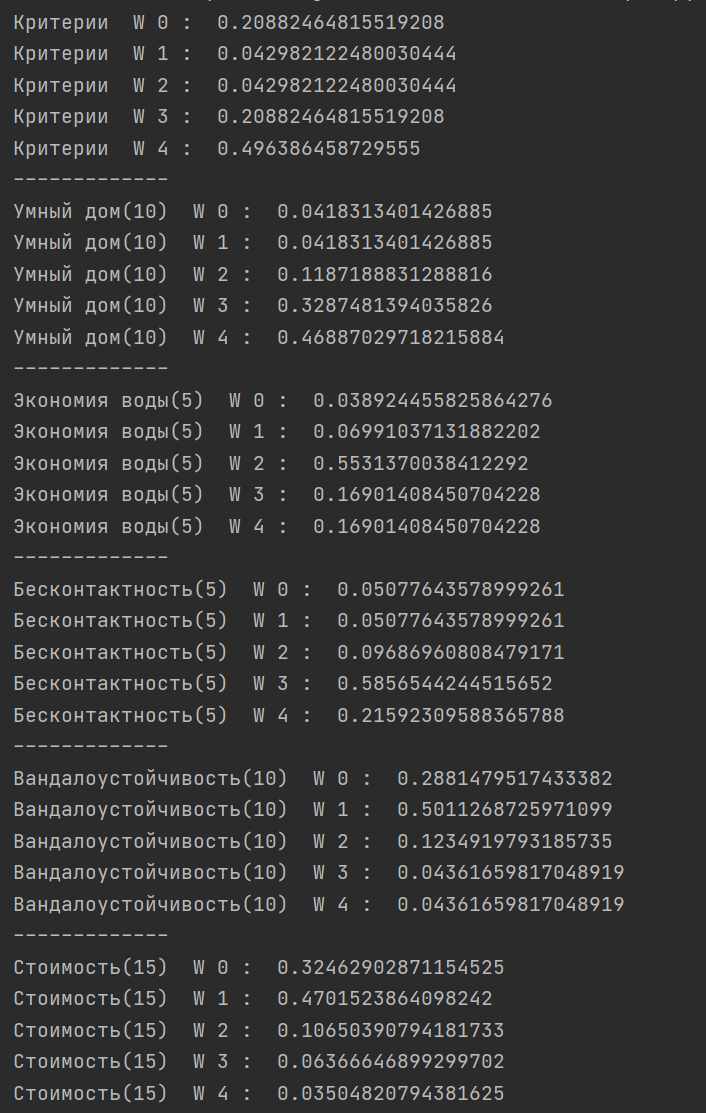


Рисунок 4. Вычисление Wi

Далее составляем из столбцов Wi каждой таблицы с альтернативами матрицу и умножаем ее на столбец из Wi для таблицы с критериями. Получаем набор чисел равный количеству альтернатив. Это оценка каждой альтернативы. Альтернатива с большим числом и есть выигрышная (рисунок 5).

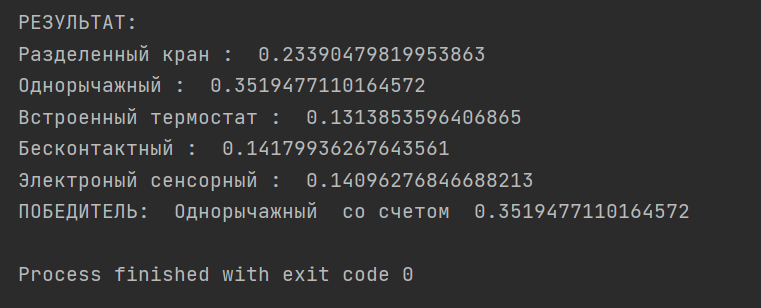


Рисунок 5. Результат работы программы

Программная реализация

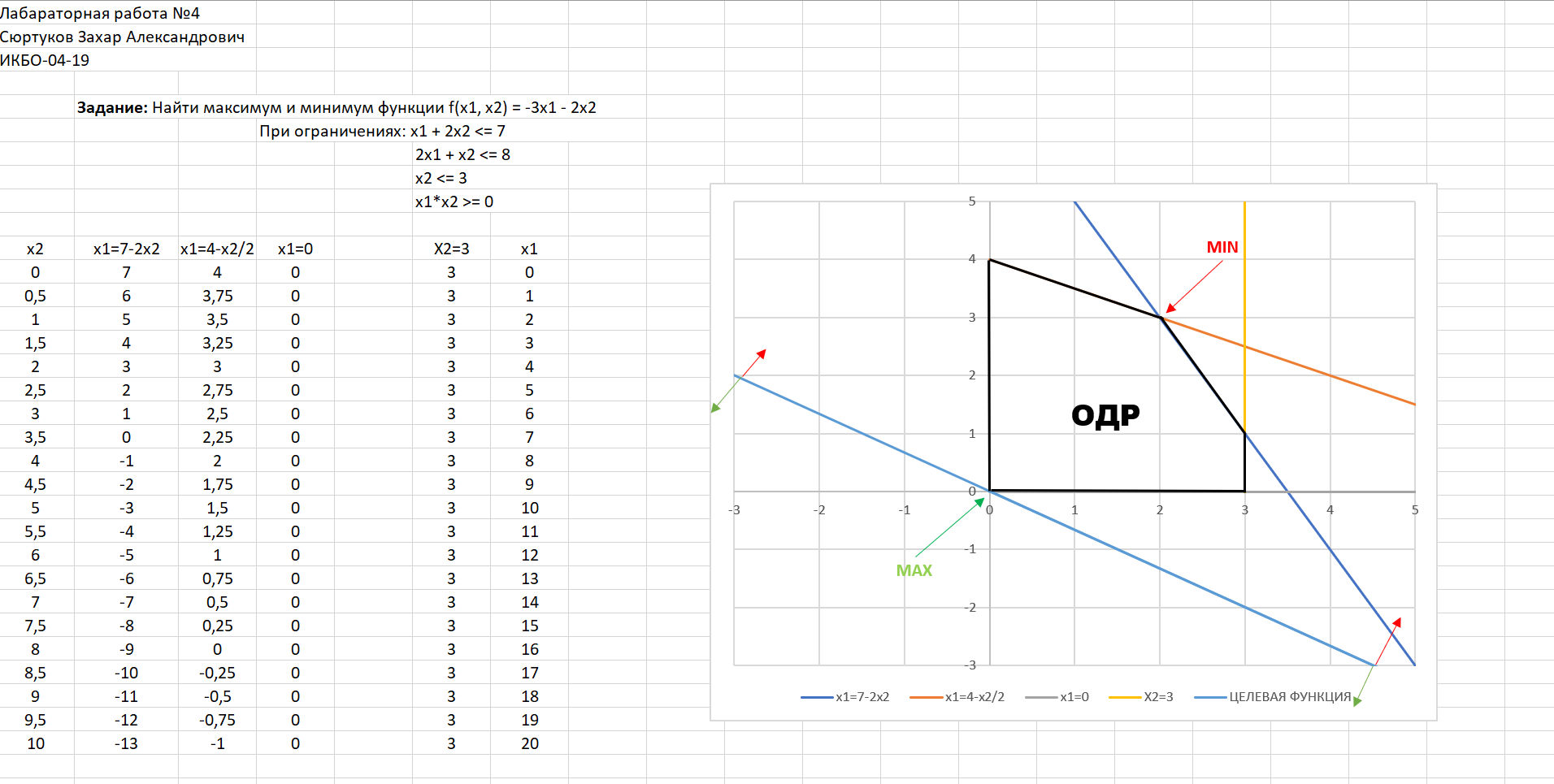
import pandas as pd  
import os  
import numpy as np  
  
from pandas import DataFrame  
  
  
def mai(path\_to\_dir, file, fileResult):  
 os.chdir(path\_to\_dir)  
 xl = pd.ExcelFile(file)  
 df1 = xl.parse(xl.sheet\_names[0])  
 criteria\_num = dict()  
 for i in range(1, len(df1) + 1):  
 criteria\_num[list(df1)[i].split("(")[0]] = (int((list(df1)[i].split("(")[1].split(")")[0])))  
 criteria\_table = pairwise\_comparison(criteria\_num, 9)  
 criteria\_matrix = np.array(compute\_w(criteria\_table, "Критерии"))  
 if not consistency\_check(criteria\_table, criteria\_matrix, 1.12):  
 return  
 print("Матрица согласована")  
 print("-------------")  
 writer = pd.ExcelWriter(fileResult, engine='xlsxwriter')  
 DataFrame(criteria\_table).to\_excel(writer, index=False)  
 start\_row = len(df1) + 2  
 alternative\_matrix\_buffer = []  
 for crit in range(1, len(df1) + 1):  
 alternative\_num = dict()  
 for alternative in range(len(df1[list(df1)[0]])):  
 alternative\_num[df1[list(df1)[0]][alternative]] = df1[list(df1)[crit]][alternative]  
 alternative\_table = pairwise\_comparison(alternative\_num, 9, list(df1)[crit])  
 DataFrame(alternative\_table).to\_excel(writer, startrow=start\_row,  
 index=False)  
 ws = compute\_w(alternative\_table, list(df1)[crit])  
 alternative\_matrix\_buffer.append(ws)  
 if not consistency\_check(alternative\_table, ws, 1.12):  
 return  
 print("Матрица согласована")  
 print("-------------")  
 start\_row += len(df1[list(df1)[0]]) + 2  
 writer.save()  
 alternative\_matrix = np.array(alternative\_matrix\_buffer).transpose()  
 result = alternative\_matrix.dot(criteria\_matrix)  
 max = -1  
 best = ''  
 print("РЕЗУЛЬТАТ:")  
 for j in range(len(result)):  
 num = result[j]  
 if num > max:  
 max = num  
 best = df1[list(df1)[0]][j]  
 print(df1[list(df1)[0]][j], ": ", num)  
 print("ПОБЕДИТЕЛЬ: ", best, " со счетом ", max)  
  
  
def compute\_w(dic, name="-"):  
 buf\_res = []  
 sum\_of\_sum = 0  
 for i in range(len(dic[list(dic)[0]])):  
 sum = 1  
 for j in range(1, len(dic)):  
 sum \*= dic[list(dic)[j]][i]  
 sum \*\*= (1/5)  
 sum = round(sum, 3)  
 buf\_res.append(sum)  
 sum\_of\_sum += sum  
 res = []  
 for i in range(len(buf\_res)):  
 buf = buf\_res[i] / sum\_of\_sum  
 print(name, " W", i, ": ", buf)  
 res.append(buf\_res[i] / sum\_of\_sum)  
 return res  
  
  
def consistency\_check(dic, ws, randIndex):  
 sum\_of\_sum = 0  
 for j in range(1, len(dic)):  
 sum = 0  
 for i in range(len(dic[list(dic)[0]])):  
 sum += dic[list(dic)[j]][i]  
 sum \*= ws[j-1]  
 print("P", j, ": ", sum)  
 sum\_of\_sum += sum  
 index = (sum\_of\_sum - len(dic) + 1)/(len(dic) - 2)  
 print("OC: ", (index / randIndex))  
 return (index / randIndex) <= 0.1  
  
  
def pairwise\_comparison(dict2, normalization, name1="-"):  
 res = {name1: []}  
 max\_pass = max(dict2.values()) - min(dict2.values())  
 for name in dict2:  
 res[name1].append(name)  
 res[name] = []  
 for i in list(dict2):  
 res[name].append(0)  
 for i in range(len(dict2)):  
 for j in range(i, len(dict2)):  
 num = round(abs(list(dict2.values())[i] - list(dict2.values())[j]) / max\_pass \* (normalization - 1)) + 1  
 if list(dict2.values())[i] > list(dict2.values())[j]:  
 res[list(res)[j + 1]][i] = num  
 res[list(res)[i + 1]][j] = 1 / num  
 else:  
 res[list(res)[i + 1]][j] = num  
 res[list(res)[j + 1]][i] = 1 / num  
 return res  
  
  
mai("C:\\Users\\zaxar\\OneDrive\\Документы\\ВУЗ\\Теория принятия решений\\МАИ", "mai.xls", "res.xlsx")

Выводы

Метод МАИ очень эффективен для решения многокритериальных задач

Линейное программирование

Графический метод



Выводы

Графический метод очень эффективен для решения задач линейного программирования

Симплекс метод

Практический расчет задачи

Первым делом мы по условию задачи составляем целевую функцию:

F(x) = 260X1 + 300X2 -> max

И ограничения:

16X1 + 12X2 <= 1200

0,2X1 + 0,4X2 <= 30

6X1 + 5X2 <= 600

3X1 + 4X2 <= 300

Следующим шагом необходимо перевести все ограничения-неравенства в ограничения-равенства путем добавления новых переменных:

16X1 + 12X2 +X3 = 1200

0,2X1 + 0,4X2 + X4 = 30

6X1 + 5X2 + X5 = 600

3X1 + 4X2 + X6 = 300

Получаем так же новую целевую функцию:

F(x) = 260X1 + 300X2 + 0X3 + 0X4 + 0X5 + 0X6 -> max

Далее вычисляем вектора для каждой переменной:

A1 = (16; 0,2; 6; 3)

A2 = (12; 0,4; 5; 4)

A3 = (1; 0; 0; 0)

A4 = (0; 1; 0; 0)

A5 = (0; 0; 1; 0)

A6 = (0; 0; 0; 1)

A0 = (1200; 30; 600; 300)

Получаем первоначальный опорный план:

X(0) = (X1, X2, X3, X4, X5, X6) = (0, 0, 1200, 30, 600, 300)

F(X(0)) = 0

Введем в рассмотрение вектор коэффициентов целевой функции при базисных переменных:

CB = (C3, C4, C5, C6) = (0, 0, 0, 0)

Найдем относительные оценки и значение целевой функции:  
&1 = CB \* A1 – C1 = -260

&2 = CB \* A2 – C2 = -300

Q = CB \* A0 = 0

Получаем изначальную симплекс таблицу:

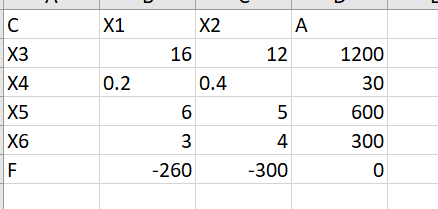


Рисунок 1. Стартовая симплекс таблица

Так как имеются отрицательные относительные оценки, необходимо пересоздать симплексную таблицу.

Число -300.0 является минимальной отрицательной относительной оценкой. Находится в столбце X2

Отношение элемента А столбца к элементу разрешающего столбца равное 75.0 является минимальным положительным отношением. Следовательно строка под номером 2 является разрешающей строкой.

Число 0.4 является разрешающим элементом

Для столбца X1 и строки под номером 1 вычисляем новый элемент через правило прямоугольника. Получаем 10.0

Для столбца X1 и строки под номером 2 вычисляем новый элемент через деление элемента на разрешающий элемент. 0.2 / 0.4 = 0.5

Для столбца X1 и строки под номером 3 вычисляем новый элемент через правило прямоугольника. Получаем 3.500000000000001

Для столбца X1 и строки под номером 4 вычисляем новый элемент через правило прямоугольника. Получаем 1.0000000000000002

Для столбца X1 и строки под номером 5 вычисляем новый элемент через правило прямоугольника. Получаем -110.0

Для столбца X2 и строки под номером 1 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 12.0 / 0.4 = -30.0

Для столбца X2 и строки под номером 2 вычисляем новый элемент через деление разрешающего элемента на единицу. 1 / 0.4 = 2.5

Для столбца X2 и строки под номером 3 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 5.0 / 0.4 = -12.5

Для столбца X2 и строки под номером 4 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 4.0 / 0.4 = -10.0

Для столбца X2 и строки под номером 5 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - -300.0 / 0.4 = 750.0

Для столбца A и строки под номером 1 вычисляем новый элемент через правило прямоугольника. Получаем 300.0

Для столбца A и строки под номером 2 вычисляем новый элемент через деление элемента на разрешающий элемент. 30 / 0.4 = 75.0

Для столбца A и строки под номером 3 вычисляем новый элемент через правило прямоугольника. Получаем 225.0

Для столбца A и строки под номером 4 вычисляем новый элемент через правило прямоугольника. Получаем 0.0

Для столбца A и строки под номером 5 вычисляем новый элемент через правило прямоугольника. Получаем 22500.0

Полученная в итоге симплекс таблица:

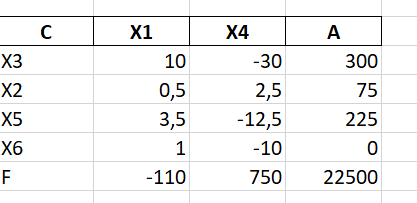


Рисунок 2. Вторая симплекс таблица

Так как имеются отрицательные относительные оценки, необходимо пересоздать симплексную таблицу.

Число -110.0 является минимальной отрицательной относительной оценкой. Находится в столбце X1

Отношение элемента А столбца к элементу разрешающего столбца равное 0.0 является минимальным положительным отношением. Следовательно строка под номером 4 является разрешающей строкой.

Число 1.0000000000000002 является разрешающим элементом

Для столбца X1 и строки под номером 1 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 10.0 / 1.0000000000000002 = -9.999999999999998

Для столбца X1 и строки под номером 2 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 0.5 / 1.0000000000000002 = -0.4999999999999999

Для столбца X1 и строки под номером 3 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 3.500000000000001 / 1.0000000000000002 = -3.5

Для столбца X1 и строки под номером 4 вычисляем новый элемент через деление разрешающего элемента на единицу. 1 / 1.0000000000000002 = 0.9999999999999998

Для столбца X1 и строки под номером 5 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - -110.0 / 1.0000000000000002 = 109.99999999999997

Для столбца X4 и строки под номером 1 вычисляем новый элемент через правило прямоугольника. Получаем 69.99999999999999

Для столбца X4 и строки под номером 2 вычисляем новый элемент через правило прямоугольника. Получаем 7.499999999999998

Для столбца X4 и строки под номером 3 вычисляем новый элемент через правило прямоугольника. Получаем 22.5

Для столбца X4 и строки под номером 4 вычисляем новый элемент через деление элемента на разрешающий элемент. -10.0 / 1.0000000000000002 = -9.999999999999998

Для столбца X4 и строки под номером 5 вычисляем новый элемент через правило прямоугольника. Получаем -349.99999999999983

Для столбца A и строки под номером 1 вычисляем новый элемент через правило прямоугольника. Получаем 300.0

Для столбца A и строки под номером 2 вычисляем новый элемент через правило прямоугольника. Получаем 75.0

Для столбца A и строки под номером 3 вычисляем новый элемент через правило прямоугольника. Получаем 225.0

Для столбца A и строки под номером 4 вычисляем новый элемент через деление элемента на разрешающий элемент. 0.0 / 1.0000000000000002 = 0.0

Для столбца A и строки под номером 5 вычисляем новый элемент через правило прямоугольника. Получаем 22500.0

Полученная симплекс таблица:

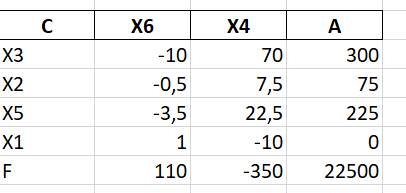


Рисунок 3. Третья симплекс таблица

Так как имеются отрицательные относительные оценки, необходимо пересоздать симплексную таблицу.

Число -349.99999999999983 является минимальной отрицательной относительной оценкой. Находится в столбце X4

Отношение элемента А столбца к элементу разрешающего столбца равное 4.2857142857142865 является минимальным положительным отношением. Следовательно строка под номером 1 является разрешающей строкой.

Число 69.99999999999999 является разрешающим элементом

Для столбца X6 и строки под номером 1 вычисляем новый элемент через деление элемента на разрешающий элемент. -9.999999999999998 / 69.99999999999999 = -0.14285714285714285

Для столбца X6 и строки под номером 2 вычисляем новый элемент через правило прямоугольника. Получаем 0.5714285714285714

Для столбца X6 и строки под номером 3 вычисляем новый элемент через правило прямоугольника. Получаем -0.28571428571428537

Для столбца X6 и строки под номером 4 вычисляем новый элемент через правило прямоугольника. Получаем -0.42857142857142866

Для столбца X6 и строки под номером 5 вычисляем новый элемент через правило прямоугольника. Получаем 59.999999999999986

Для столбца X4 и строки под номером 1 вычисляем новый элемент через деление разрешающего элемента на единицу. 1 / 69.99999999999999 = 0.014285714285714289

Для столбца X4 и строки под номером 2 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 7.499999999999998 / 69.99999999999999 = -0.10714285714285714

Для столбца X4 и строки под номером 3 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - 22.5 / 69.99999999999999 = -0.3214285714285715

Для столбца X4 и строки под номером 4 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - -9.999999999999998 / 69.99999999999999 = 0.14285714285714285

Для столбца X4 и строки под номером 5 вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 - -349.99999999999983 / 69.99999999999999 = 4.999999999999998

Для столбца A и строки под номером 1 вычисляем новый элемент через деление элемента на разрешающий элемент. 300.0 / 69.99999999999999 = 4.2857142857142865

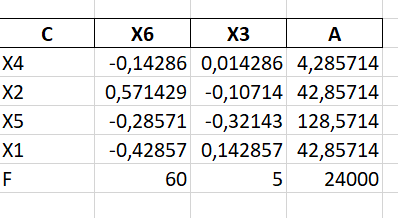
Для столбца A и строки под номером 2 вычисляем новый элемент через правило прямоугольника. Получаем 42.85714285714286

Для столбца A и строки под номером 3 вычисляем новый элемент через правило прямоугольника. Получаем 128.57142857142856

Для столбца A и строки под номером 4 вычисляем новый элемент через правило прямоугольника. Получаем 42.85714285714286

Для столбца A и строки под номером 5 вычисляем новый элемент через правило прямоугольника. Получаем 24000.0

Полученная симплекс таблица:



Т. к. нет отрицательных относительных оценок таблица является окончательной.

X(3) = (X1 = 42,86; X2 = 42,86; X3 = 0; X4 = 4,28; X5 = 128,57)

Fmax = f(X(3)) = 42,86 \* 260 + 42,86 \* 300 = 24000

Проверим выполнение ограничений:

16\*42,86 + 12\*42,86 <= 1200

0,2\*42,86 + 0,4\*42,86 <= 30

6\*42,86 + 5\*42,86 <= 600

3\*42,86 + 4\*42,86 <= 300

Программная реализация

import pandas as pd  
import numpy as np  
  
def simplex\_recurs(df):  
 # Вычисляем разрешающий столбец  
 print(df)  
 print()  
 min = 0  
 column = ""  
 for i in range(1, len(df) - 2):  
 key = list(df)[i]  
 mass = df[key]  
 if mass[len(mass) - 1] < min:  
 min = mass[len(mass) - 1]  
 column = key  
 print("Число", min, "является минимальной отрицательной относительной оценкой. Находится в столбце", column)  
 print()  
 # Вычисляем разрешающую строку  
 massA = df["A"]  
 massColumn = df[column]  
 raw = 0  
 min = -1  
 for i in range(len(massA) - 1):  
 if float(massColumn[i]) <= 0:  
 continue  
 num = float(massA[i]) / float(massColumn[i])  
 if min == -1 or num < min:  
 min = num  
 raw = i  
 print("Отношение элемента А столбца к элементу разрешающего столбца равное", min, "является минимальным "  
 "положительным отношением. "  
 "Следовательно строка под "  
 "номером", raw + 1,  
 "является разрешающей строкой.")  
 print()  
 if min == -1:  
 raise Exception("Целевая функция не ограничена")  
 resolution\_element = df[column][raw]  
 print("Число", resolution\_element, "является разрешающим элементом")  
 print()  
 # column - ключ разрещающего столбца, raw - номер разрещающей строки, resolution\_element - разрещающий элемент  
 # Формируем новую таблицу  
 newDf = dict()  
 # Заолняем первый столбец  
 newDf["C"] = []  
 for i in range(len(massA) - 1):  
 if i == raw:  
 newDf["C"].append(column)  
 else:  
 newDf["C"].append(df["C"][i])  
 newDf["C"].append("F")  
 final = True  
 # Заполняем остальные строки  
 for key in df:  
 if key == "C":  
 continue  
 # Отдельно заполняем разрещающий столбец  
 if key == column:  
 newDf[df["C"][raw]] = []  
 for i in range(len(df[key])):  
 # Отдельно заполняем разрещающий элемент  
 if i == raw:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через деление разрешающего элемента на единицу. 1 /",  
 df[key][i], " =", 1 / df[key][i])  
 print()  
 newDf[df["C"][raw]].append(1 / df[key][i])  
 else:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 -",  
 df[key][i], " /", resolution\_element, " =", 0 - df[key][i] / resolution\_element)  
 print()  
 newDf[df["C"][raw]].append(0 - df[key][i] / resolution\_element)  
 if i == len(df[key]) - 1 and newDf[df["C"][raw]][i] < 0:  
 final = False  
 else:  
 newDf[key] = []  
 for i in range(len(df[key])):  
 # Отдельно заполняем разрещающую строку  
 if i == raw:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через деление элемента на разрешающий элемент.",  
 df[key][i], " /", resolution\_element, " =", df[key][i] / resolution\_element)  
 print()  
 newDf[key].append(df[key][i] / resolution\_element)  
 # Заполняем все остальные элементы через прямоугольник  
 else:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через правило прямоугольника. Получаем ",  
 (df[key][i] \* resolution\_element - df[column][i] \* df[key][raw]) / resolution\_element)  
 print()  
 newDf[key].append(  
 (df[key][i] \* resolution\_element - df[column][i] \* df[key][raw]) / resolution\_element)  
 if i == len(df[key]) - 1 and key != "A" and newDf[key][i] < 0:  
 final = False  
 res = [pd.DataFrame(newDf)]  
 if final:  
 return res  
 # Если остались отрицательные дельты, то продолжаем выполнение  
 print()  
 print()  
 print("------------------------------------------")  
 print()  
 print()  
 res.extend(simplex\_recurs(pd.DataFrame(newDf)))  
 return res  
  
  
def dual\_task(df, basis\_list):  
 for i in range(5):  
 print()  
 print("ДВОЙСТВЕННАЯ ЗАДАЧА")  
 target = ""  
 limitations = []  
 # Заносим правую часть неравенства для ограничений  
 for i in range(len(df) - 3):  
 limitations.append(str(0 - df["X" + str(i + 1)][len(df["X" + str(i + 1)]) - 1]) + " <= ")  
 for i in range(len(df["A"]) - 1):  
 if i != 0:  
 target += " + "  
 # Создаем целевую функцию  
 target += str(df["A"][i]) + "y" + str((i + 1))  
 # Создаем ограничения  
 for j in range(len(df) - 3):  
 if i != 0:  
 limitations[j] += " + "  
 limitations[j] += str(df["X" + str(j + 1)][i]) + "y" + str((i + 1))  
 print("Целевая функция: ")  
 print(target)  
 print("Ограничения: ")  
 for i in range(len(df) - 3):  
 print(limitations[i])  
 # Создаем матрицу обратную D матрицу из векторов A  
 D = []  
 for x in basis\_list:  
 try:  
 res = df[x].tolist()  
 res.pop(len(res) - 1)  
 D.append(res)  
 except Exception:  
 res = [0 for i in range(len(basis\_list))]  
 res[int(x[1]) - (len(df) - 2)] = 1  
 D.append(res)  
 print("Матрица D:")  
 print(np.array(D).transpose())  
 Drev = np.linalg.inv(np.array(D).transpose())  
 print("Обратная D матрица:")  
 print(Drev)  
 # Создаем вектор Cb из коэффициентов C  
 Cb = []  
 for x in basis\_list:  
 try:  
 Cb.append(0 - df[x][len(basis\_list)])  
 except Exception:  
 Cb.append(0)  
 print("Вектор Cb:")  
 print(Cb)  
 # Получаем вектор y  
 y = np.array(Cb).dot(Drev)  
 print("Вектор y:")  
 print(y)  
 res = 0  
 for i in range(len(df["A"]) - 1):  
 res += df["A"][i] \* y[i]  
 print("Значение целевой функции:")  
 print(round(res, 0))  
  
  
def simplex(fileStart, fileResult):  
 xl = pd.ExcelFile(fileStart)  
 df1 = xl.parse(xl.sheet\_names[0])  
 res = simplex\_recurs(df1)  
 writer = pd.ExcelWriter(fileResult, engine='xlsxwriter')  
 df1.to\_excel(writer, index=False)  
 start\_row = len(df1) + 2  
 for df in res:  
 df.to\_excel(writer, startrow=start\_row,  
 index=False)  
 start\_row += len(df) + 2  
 basis\_list = res[len(res) - 1]["C"]  
 basis\_list.pop(len(basis\_list) - 1)  
 dual\_task(df1, res[len(res) - 1]["C"])  
 writer.save()  
  
  
simplex("Starting.xls", "Result.xlsx")

Выводы

Симплексный метод оказался очень эффективен для решения задач линейного программирования

Двойственная задача

Практический расчет задачи

Ход решения 1 теоремы двойственности

Для примера возьмем задачу, которая решалась симплексным методом. Вспомним ее целевую функцию и ограничения:

F(x) = 260X1 + 300X2 -> max

16X1 + 12X2 <= 1200

0,2X1 + 0,4X2 <= 30

6X1 + 5X2 <= 600

3X1 + 4X2 <= 300

Составим математическую модель двойственной задачи. В качестве переменных двойственной задачи возьмем y1, y2, y3, y4. Так как данная задача является симметричной, двойственная задача в матричном виде будет выглядеть следующим образом:

g(y) = (b, y) -> min

Atrans \* y => c

y > 0

Atrans = - транспонированная матрица коэффициентов системы ограничений.

Необходимо найти минимум целевой функции:

g(y) = (b, y) = 1200y1 + 30y2 + 600y3 + 300y4

При следующих ограничениях:

16y1 + 0,2y2 + 6y3 + 3y4 => 260

12y1 + 0,4y2 + 5y3 + 4y4 => 300

Все “y” больше 0.

Решение может быть найдено из формулы:

g(y) = Cb \* Drev

где Drev – обратная матрица, составленная из компонентов векторов, входящих в последний базис, при котором получен оптимальный план исходной задачи. В нашем примере базисными переменными являлись x4, x2, x5, x1. Этим переменным соответствуют векторы:

A4 = (0; 1; 0; 0)

A2 = (12; 0,4; 5; 4)

A5 = (0; 0; 1; 0)

A1 = (16; 0,2; 6; 3)

Тогда:

D =

Найдем обратную матрицу Drev:

Drev =

Cb = (0, 300, 0, 260)

Получаем:

y = Cb \* Drev = (5, 0, 0, 60)

Подставляем:

g(y) = 24000

Таким образом:

max f(x) = min g(y) = 24000

Ход решения 2 теоремы двойственности

Итак, имеем оптимальное решение прямой задачи:

X1 = 42.857

X2 = 42.857

Fmax = 24000

Рассмотрим выполнение неравенств прямой задачи при подстановке в систему ограничений:

1. Ограничение: 16x1 + 12x2 <= 1200

Подстановка: 16\*42.857 + 12\*42.857 = 1200

Вывод: ресурс является дефицитным и его оценка отлична от нуля (y1 != 0)

1. Ограничение: 0.2\*x1 + 0.4\*x2 <= 30

Подстановка: 0.2\*42.857 + 0.4\*42.857 = 25

Вывод: ресурс не является дефицитным и его оценка равна нулю (y2 = 0)

1. Ограничение: 6\*x1 + 5\*x2 <= 600

Подстановка: 6\*42.857 + 5\*42.857 = 462

Вывод: ресурс не является дефицитным и его оценка равна нулю (y3 = 0)

1. Ограничение: 3\*x1 + 4\*x2 <= 300

Подстановка: 3\*42.857 + 4\*42.857 = 300

Вывод: ресурс является дефицитным и его оценка отлична от нуля (y4 != 0)

Ход решения 3 теоремы двойственности

Оценим чувствительность решения задачи о максимальном доходе к изменению запасов сырья и спроса на продукцию. Вспомним, что оптимальное решение двойственной задачи равно:

Y = (5, 0, 0, 60)

Предположим, что спрос увеличился на единицу:

Gmax1 = y1 \* b1 = 5 \* 1 = 5

Gmax4 = y4 \* b4 = 60 \* 1 = 60

Обратная матрица базиса оптимального плана:

Drev =

Индексы базисных переменных оптимального плана:

A0\* = (0, 300, 0, 260)

Свободные члены неравенств прямой задачи:

A0 = (1200, 30, 600, 300)

Программная реализация

import pandas as pd  
import numpy as np  
  
def simplex\_recurs(df):  
 # Вычисляем разрешающий столбец  
 print(df)  
 print()  
 min = 0  
 column = ""  
 for i in range(1, len(df) - 2):  
 key = list(df)[i]  
 mass = df[key]  
 if mass[len(mass) - 1] < min:  
 min = mass[len(mass) - 1]  
 column = key  
 print("Число", min, "является минимальной отрицательной относительной оценкой. Находится в столбце", column)  
 print()  
 # Вычисляем разрешающую строку  
 massA = df["A"]  
 massColumn = df[column]  
 raw = 0  
 min = -1  
 for i in range(len(massA) - 1):  
 if float(massColumn[i]) <= 0:  
 continue  
 num = float(massA[i]) / float(massColumn[i])  
 if min == -1 or num < min:  
 min = num  
 raw = i  
 print("Отношение элемента А столбца к элементу разрешающего столбца равное", min, "является минимальным "  
 "положительным отношением. "  
 "Следовательно строка под "  
 "номером", raw + 1,  
 "является разрешающей строкой.")  
 print()  
 if min == -1:  
 raise Exception("Целевая функция не ограничена")  
 resolution\_element = df[column][raw]  
 print("Число", resolution\_element, "является разрешающим элементом")  
 print()  
 # column - ключ разрещающего столбца, raw - номер разрещающей строки, resolution\_element - разрещающий элемент  
 # Формируем новую таблицу  
 newDf = dict()  
 # Заолняем первый столбец  
 newDf["C"] = []  
 for i in range(len(massA) - 1):  
 if i == raw:  
 newDf["C"].append(column)  
 else:  
 newDf["C"].append(df["C"][i])  
 newDf["C"].append("F")  
 final = True  
 # Заполняем остальные строки  
 for key in df:  
 if key == "C":  
 continue  
 # Отдельно заполняем разрещающий столбец  
 if key == column:  
 newDf[df["C"][raw]] = []  
 for i in range(len(df[key])):  
 # Отдельно заполняем разрещающий элемент  
 if i == raw:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через деление разрешающего элемента на единицу. 1 /",  
 df[key][i], " =", 1 / df[key][i])  
 print()  
 newDf[df["C"][raw]].append(1 / df[key][i])  
 else:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через деление элемента на разрешающий элемент и изменение знака. 0 -",  
 df[key][i], " /", resolution\_element, " =", 0 - df[key][i] / resolution\_element)  
 print()  
 newDf[df["C"][raw]].append(0 - df[key][i] / resolution\_element)  
 if i == len(df[key]) - 1 and newDf[df["C"][raw]][i] < 0:  
 final = False  
 else:  
 newDf[key] = []  
 for i in range(len(df[key])):  
 # Отдельно заполняем разрещающую строку  
 if i == raw:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через деление элемента на разрешающий элемент.",  
 df[key][i], " /", resolution\_element, " =", df[key][i] / resolution\_element)  
 print()  
 newDf[key].append(df[key][i] / resolution\_element)  
 # Заполняем все остальные элементы через прямоугольник  
 else:  
 print("Для столбца", key, "и строки под номером", i + 1,  
 "вычисляем новый элемент через правило прямоугольника. Получаем ",  
 (df[key][i] \* resolution\_element - df[column][i] \* df[key][raw]) / resolution\_element)  
 print()  
 newDf[key].append(  
 (df[key][i] \* resolution\_element - df[column][i] \* df[key][raw]) / resolution\_element)  
 if i == len(df[key]) - 1 and key != "A" and newDf[key][i] < 0:  
 final = False  
 res = [pd.DataFrame(newDf)]  
 if final:  
 return res  
 # Если остались отрицательные дельты, то продолжаем выполнение  
 print()  
 print()  
 print("------------------------------------------")  
 print()  
 print()  
 res.extend(simplex\_recurs(pd.DataFrame(newDf)))  
 return res  
  
  
def dual\_task(df, basis\_list):  
 for i in range(5):  
 print()  
 print("ДВОЙСТВЕННАЯ ЗАДАЧА")  
 target = ""  
 limitations = []  
 # Заносим правую часть неравенства для ограничений  
 for i in range(len(df) - 3):  
 limitations.append(str(0 - df["X" + str(i + 1)][len(df["X" + str(i + 1)]) - 1]) + " <= ")  
 for i in range(len(df["A"]) - 1):  
 if i != 0:  
 target += " + "  
 # Создаем целевую функцию  
 target += str(df["A"][i]) + "y" + str((i + 1))  
 # Создаем ограничения  
 for j in range(len(df) - 3):  
 if i != 0:  
 limitations[j] += " + "  
 limitations[j] += str(df["X" + str(j + 1)][i]) + "y" + str((i + 1))  
 print("Целевая функция: ")  
 print(target)  
 print("Ограничения: ")  
 for i in range(len(df) - 3):  
 print(limitations[i])  
 # Создаем матрицу обратную D матрицу из векторов A  
 D = []  
 for x in basis\_list:  
 try:  
 res = df[x].tolist()  
 res.pop(len(res) - 1)  
 D.append(res)  
 except Exception:  
 res = [0 for i in range(len(basis\_list))]  
 res[int(x[1]) - (len(df) - 2)] = 1  
 D.append(res)  
 print("Матрица D:")  
 print(np.array(D).transpose())  
 Drev = np.linalg.inv(np.array(D).transpose())  
 print("Обратная D матрица:")  
 print(Drev)  
 # Создаем вектор Cb из коэффициентов C  
 Cb = []  
 for x in basis\_list:  
 try:  
 Cb.append(0 - df[x][len(basis\_list)])  
 except Exception:  
 Cb.append(0)  
 print("Вектор Cb:")  
 print(Cb)  
 # Получаем вектор y  
 y = np.array(Cb).dot(Drev)  
 print("Вектор y:")  
 print(y)  
 res = 0  
 for i in range(len(df["A"]) - 1):  
 res += df["A"][i] \* y[i]  
 print("Значение целевой функции:")  
 print(round(res, 0))  
  
  
def simplex(fileStart, fileResult):  
 xl = pd.ExcelFile(fileStart)  
 df1 = xl.parse(xl.sheet\_names[0])  
 res = simplex\_recurs(df1)  
 writer = pd.ExcelWriter(fileResult, engine='xlsxwriter')  
 df1.to\_excel(writer, index=False)  
 start\_row = len(df1) + 2  
 for df in res:  
 df.to\_excel(writer, startrow=start\_row,  
 index=False)  
 start\_row += len(df) + 2  
 basis\_list = res[len(res) - 1]["C"]  
 basis\_list.pop(len(basis\_list) - 1)  
 dual\_task(df1, res[len(res) - 1]["C"])  
 writer.save()  
  
  
simplex("Starting.xls", "Result.xlsx")

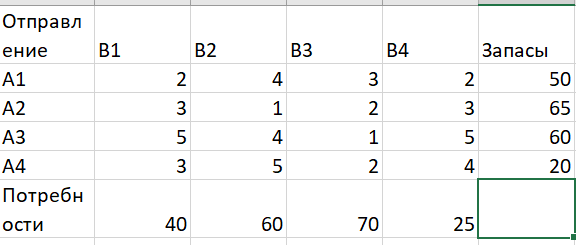
Выводы

Двойственная задача очень эффективна при решении задач линейного программирования

Транспортная задача

Практический расчет задачи

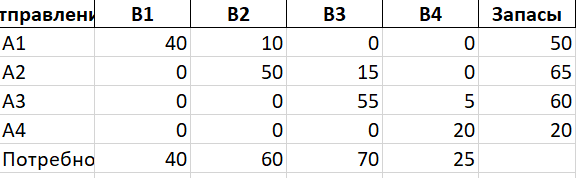
Берем стартовую таблицу с потребностями, ресурсами и значениями C:



Применяем метод северо-западного узла для создания опорного плана.

1. На потребность B1 выделяем 40 единиц из A1
2. На потребность B2 выделяем оставшиеся 10 единиц A1
3. На потребность B2 выделяем необходимые 50 единиц из А2
4. На потребность B3 выделяем оставшиеся 15 единиц из A2
5. На потребность B3 выделяем необходимые 55 единиц из A3
6. На потребность B4 выделяем оставшиеся 5 единиц из A3
7. На потребность B4 оставшиеся и необходимые 20 единиц из A4

Получаем:



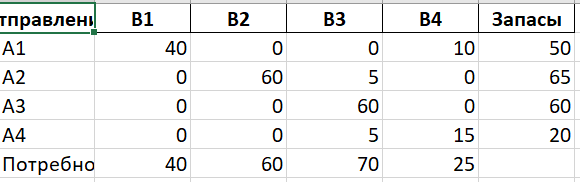
Сумма этой таблицы:

F = 360

Применяем метод минимальной стоимости для создания опорного плана:

1. На потребность B2 для полного заполнения выделяем 60 единиц из A2
2. На потребность B3 выделяем оставшиеся в A3 60 единиц
3. На потребность В1 для полного заполнения выделяем 40 единиц из А1
4. На потребность B3 выделяем оставшиеся в A2 5 единиц
5. На потребность B3 для полного заполнения выделяем 5 единиц из A4
6. На потребность B4 выделяем оставшиеся в A1 10 единиц
7. На потребность B4 для полного заполнения выделяем 15 единиц из A4

Получаем:



Сумма этой таблицы:

F = 300

Для большей наглядности возьмем за опорный план таблицу, полученную методом северо-западного узла

Дальше в дело вступает метод потенциалов. Для начала вычислим потенциалы. Для этого составим уравнения через значения C в изначальной таблице:

U0 = 0

U0 + Y0 = 2

Y0 = 2

U0 + Y1 = 4

Y1 = 4

U1 + Y1 = 1

U1 = -3

U1 + Y2 = 2

Y2 = 5

U2 + Y2 = 1

U2 = -4

U2 + Y3 = 5

Y3 = 9

U3 + Y3 = 4

U3 = -5

Зная все потенциалы, вычисляем относительные оценки и находим минимальную:

V10 = 4

V20 = 7

V30 = 6

V21 = 4

V31 = 6

V02 = -2

V32 = 2

V03 = -7

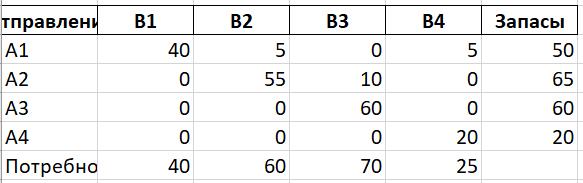
V13 = -3

min: -7

Минимальная оценка на координатах 3 и 0. Проводим цикл начиная с этой точки. Получаем:

[[3, 0], [1, 0], [1, 1], [2, 1], [2, 2], [3, 2]]

Минимальное значение 5 на позиции [3, 2]. Проводим уменьшение и увеличение вершин на это значение. Получаем:



Это новая таблица. Ее сумма:

F = 165

Для начала вычислим потенциалы. Для этого составим уравнения через значения C в изначальной таблице:

U0 = 0

U0 + Y0 = 2

Y0 = 2

U0 + Y3 = 2

Y3 = 2

U0 + Y1 = 4

Y1 = 4

U3 + Y3 = 4

U3 = 2

U1 + Y1 = 1

U1 = -3

U1 + Y2 = 2

Y2 = 5

U2 + Y2 = 1

U2 = -4

Зная все потенциалы, вычисляем относительные оценки и находим минимальную:

V10 = 4

V20 = 7

V30 = -1

V21 = 4

V31 = -1

V02 = -2

V32 = -5

V13 = 4

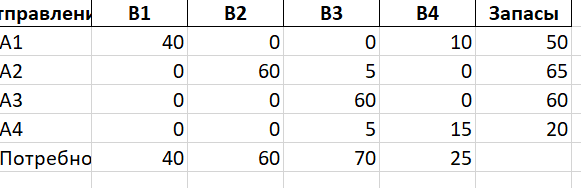
V23 = 7

min: -5

Минимальная оценка на координатах 2 и 3. Проводим цикл начиная с этой точки. Получаем:

[[2, 3], [3, 3], [3, 0], [1, 0], [1, 1], [2, 1]]

Минимальное значение 5 на позиции [1, 0]. Проводим уменьшение и увеличение вершин на это значение. Получаем:



Это новая таблица. Ее сумма:

F = 160

Для начала вычислим потенциалы. Для этого составим уравнения через значения C в изначальной таблице:

U0 = 0

U0 + Y0 = 2

Y0 = 2

U0 + Y3 = 2

Y3 = 2

U3 + Y3 = 4

U3 = 2

U3 + Y2 = 2

Y2 = 0

U1 + Y2 = 2

U1 = 2

U1 + Y1 = 1

Y1 = -1

U2 + Y2 = 1

U2 = 1

Зная все потенциалы, вычисляем относительные оценки и находим минимальную:

V10 = -1

V20 = 2

V30 = -1

V01 = 5

V21 = 4

V31 = 4

V02 = 3

V13 = -1

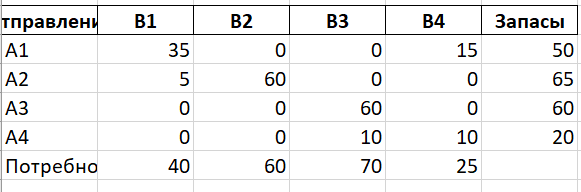
V23 = 2

min: -1

Минимальная оценка на координатах 0 и 1. Проводим цикл начиная с этой точки. Получаем:

[[0, 1], [2, 1], [2, 3], [3, 3], [3, 0], [0, 0]]

Минимальное значение 5 на позиции [2, 3]. Проводим уменьшение и увеличение вершин на это значение. Получаем:



Это новая таблица. Ее сумма:

F = 155

Для начала вычислим потенциалы. Для этого составим уравнения через значения C в изначальной таблице:

U0 = 0

U0 + Y0 = 2

Y0 = 2

U0 + Y3 = 2

Y3 = 2

U1 + Y0 = 3

U1 = 1

U1 + Y1 = 1

Y1 = 0

U3 + Y3 = 4

U3 = 2

U3 + Y2 = 2

Y2 = 0

U2 + Y2 = 1

U2 = 1

Зная все потенциалы, вычисляем относительные оценки и находим минимальную:

V20 = 2

V30 = -1

V01 = 4

V21 = 3

V31 = 3

V02 = 3

V12 = 1

V13 = 0

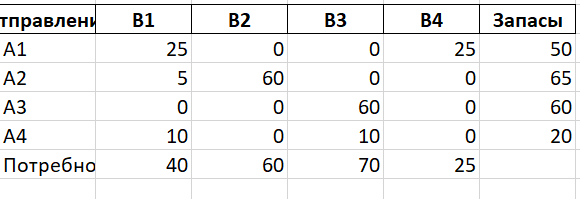
V23 = 2

min: -1

Минимальная оценка на координатах 0 и 3. Проводим цикл начиная с этой точки. Получаем:

[[0, 3], [3, 3], [3, 0], [0, 0]]

Минимальное значение 10 на позиции [3, 3]. Проводим уменьшение и увеличение вершин на это значение. Получаем:



Это новая таблица. Ее сумма:

F = 130

Для начала вычислим потенциалы. Для этого составим уравнения через значения C в изначальной таблице:

U0 = 0

U0 + Y0 = 2

Y0 = 2

U0 + Y3 = 2

Y3 = 2

U1 + Y0 = 3

U1 = 1

U1 + Y1 = 1

Y1 = 0

U3 + Y0 = 3

U3 = 1

U3 + Y2 = 2

Y2 = 1

U2 + Y2 = 1

U2 = 0

Зная все потенциалы, вычисляем относительные оценки и находим минимальную:

V20 = 3

V01 = 4

V21 = 4

V31 = 4

V02 = 2

V12 = 0

V13 = 0

V23 = 3

V33 = 1

min: 0

Минимальное значение неотрицательное. Значит мы смогли получить оптимальное решение!

Программная реализация

import pandas as pd  
import sys  
  
  
def transport\_task(df):  
 # Создаем списки запасов и требований узлов  
 stocks = list(df["Запасы"])  
 stocks.pop()  
 num\_of\_needs = len(df["Запасы"]) - 1  
 needs = []  
 reference\_plan = df.copy(deep=True)  
 for i in range(1, len(df)):  
 needs.append(df["B" + str(i)][num\_of\_needs])  
 for j in range(num\_of\_needs):  
 reference\_plan.loc[j, "B" + str(i)] = 0  
 # Применяем метод северо-западного узла  
 reference\_plan\_north = reference\_plan.copy(deep=True)  
 reference\_plan\_minmax = reference\_plan.copy(deep=True)  
 equations\_plan\_north = []  
 equations\_plan\_minmax = []  
 min\_north = northwest\_corner(reference\_plan\_north, equations\_plan\_north, df, stocks.copy(), needs.copy())  
 min\_min\_max = min\_max(reference\_plan\_minmax, equations\_plan\_minmax, df, stocks.copy(), needs.copy())  
 print("Опорный план северо-западного узла")  
 print(reference\_plan\_north)  
 print("F(северо-западный узел) =", min\_north)  
 print()  
 print("Опорный план минимальной стоимости")  
 print(reference\_plan\_minmax)  
 print("F(минимальная стоимость) =", min\_min\_max)  
 print()  
 writer = pd.ExcelWriter("Result.xlsx", engine='xlsxwriter')  
 if len(equations\_plan\_minmax) != (len(needs) + len(stocks) - 1):  
 print("План минимальной стоимости является вырожденным, выбираем северо-западный узел")  
 recurs\_transport(df, reference\_plan\_north, equations\_plan\_north, writer, 0)  
 else:  
 if min\_north <= min\_min\_max:  
 print("План северо-западного узла является более оптимальным, выбираем его")  
 recurs\_transport(df, reference\_plan\_north, equations\_plan\_north, writer, 0)  
 else:  
 print("План минимальной стоимости является более оптимальным, выбираем его")  
 recurs\_transport(df, reference\_plan\_minmax, equations\_plan\_minmax, writer, 0)  
  
  
def northwest\_corner(reference\_plan, equations, df, stocks, needs):  
 sum = 0  
 for i in range(len(stocks)):  
 for j in range(len(needs)):  
 if needs[j] != 0:  
 equations.append([i, j, df.loc[i, "B" + str(j + 1)]])  
 if stocks[i] < needs[j]:  
 reference\_plan.loc[i, "B" + str(j + 1)] = stocks[i]  
 needs[j] = needs[j] - stocks[i]  
 stocks[i] = 0  
 sum += reference\_plan.loc[i, "B" + str(j + 1)] \* df.loc[i, "B" + str(j + 1)]  
 break  
 else:  
 reference\_plan.loc[i, "B" + str(j + 1)] = needs[j]  
 stocks[i] = stocks[i] - needs[j]  
 needs[j] = 0  
 sum += reference\_plan.loc[i, "B" + str(j + 1)] \* df.loc[i, "B" + str(j + 1)]  
 return sum  
  
  
def min\_max(reference\_plan, equations, df, stocks, needs):  
 null\_needs = 0  
 sum = 0  
 while null\_needs != len(needs):  
 min\_c = sys.maxsize  
 cords = []  
 for i in range(1, len(df)):  
 for j in range(len(df["B" + str(i)]) - 1):  
 if needs[i - 1] != 0 and stocks[j] != 0 and df.loc[j, "B" + str(i)] < min\_c:  
 min\_c = df.loc[j, "B" + str(i)]  
 cords.clear()  
 cords.append(j)  
 cords.append(i - 1)  
 equations.append([cords[0], cords[1], df.loc[cords[0], "B" + str(cords[1] + 1)]])  
 if stocks[cords[0]] < needs[cords[1]]:  
 needs[cords[1]] = needs[cords[1]] - stocks[cords[0]]  
 reference\_plan.loc[cords[0], "B" + str(cords[1] + 1)] = stocks[cords[0]]  
 stocks[cords[0]] = 0  
 else:  
 stocks[cords[0]] = stocks[cords[0]] - needs[cords[1]]  
 reference\_plan.loc[cords[0], "B" + str(cords[1] + 1)] = needs[cords[1]]  
 needs[cords[1]] = 0  
 null\_needs += 1  
 sum += reference\_plan.loc[cords[0], "B" + str(cords[1] + 1)] \* df.loc[cords[0], "B" + str(cords[1] + 1)]  
 return sum  
  
  
def recurs\_transport(df, table, equations, writer, step):  
 print("------------------------")  
 table.to\_excel(writer, startrow=step, index=False)  
 min\_v = solving\_equations(equations.copy(), df, table)  
 # Если нет отрицательных оценок, то завершаем выполнение  
 if min\_v[2] >= 0:  
 print("Получено оптимальное решение!")  
 df.to\_excel(writer, startrow=(step + 10), index=False)  
 writer.save()  
 return  
 # Получаем транспонированную матрицу значений  
 matrix = []  
 for i in range(1, len(table)):  
 buf = list(table["B" + str(i)])  
 buf.pop()  
 matrix.append(buf)  
 vertexes = precycle(matrix, min\_v[1], min\_v[0])  
 min = sys.maxsize  
 # Ищем минимальное значение среди вершин  
 print()  
 print(vertexes)  
 for i in range(1, len(vertexes), 2):  
 vertex = vertexes[i]  
 num = table.loc[vertex[1], "B" + str(vertex[0] + 1)]  
 if num < min and num != 0:  
 min = num  
 # Уменьшаем и увеличиваем вершины в цикле  
 print("Минимальное значение:", min)  
 sum = 0  
 for vertex in vertexes:  
 table.loc[vertex[1], "B" + str(vertex[0] + 1)] = table.loc[vertex[1], "B" + str(vertex[0] + 1)] + min  
 sum += df.loc[vertex[1], "B" + str(vertex[0] + 1)] \* table.loc[vertex[1], "B" + str(vertex[0] + 1)]  
 # Удаляем нулевые уравнений  
 if table.loc[vertex[1], "B" + str(vertex[0] + 1)] == 0:  
 for equation in equations:  
 if equation[0] == vertex[1] and equation[1] == vertex[0]:  
 equations.remove(equation)  
 min = 0 - min  
 # Добавляем новое уравнение  
 equations.append([min\_v[0], min\_v[1], df.loc[min\_v[0], "B" + str(min\_v[1] + 1)]])  
 print()  
 print(table)  
 print()  
 print("F =", sum)  
 print()  
 recurs\_transport(df, table, equations, writer, step + 10)  
  
  
def precycle(matrix, y, x):  
 cords = [[y, x]]  
 # Проверяем горизонтальное и вертикальное направления  
 if not cycle(matrix, cords, True):  
 cycle(matrix, cords, False)  
 return cords  
  
  
def cycle(matrix, cords, vertical):  
 final\_cord = cords[0]  
 cord = cords[len(cords) - 1]  
 if vertical:  
 x = cord[1]  
 # Проверяем на возвращение в исходную точку  
 if len(cords) != 1 and x == final\_cord[1]:  
 return True  
 # Собираем y, которые уже были с этим x  
 ys = []  
 for old\_cord in cords:  
 if old\_cord[1] == x:  
 ys.append(old\_cord[0])  
 for y in range(len(matrix)):  
 # Ищем новые вершины  
 if not ys.\_\_contains\_\_(y) and matrix[y][x] != 0:  
 cords.append([y, x])  
 if cycle(matrix, cords, False):  
 return True  
 cords.pop()  
 else:  
 y = cord[0]  
 # Проверяем на возвращение в исходную точку  
 if len(cords) != 1 and y == final\_cord[0]:  
 return True  
 # Собираем x, которые уже были с этим y  
 xs = []  
 for old\_cord in cords:  
 if old\_cord[0] == y:  
 xs.append(old\_cord[1])  
 for x in range(len(matrix[0])):  
 # Ищем новые вершины  
 if not xs.\_\_contains\_\_(x) and matrix[y][x] != 0:  
 cords.append([y, x])  
 if cycle(matrix, cords, True):  
 return True  
 cords.pop()  
 return False  
  
  
def solving\_equations(equations, df, table):  
 # Решаем систему уравнений  
 min = sys.maxsize  
 res = []  
 u = [sys.maxsize for i in range(len(df) - 1)]  
 y = [sys.maxsize for i in range(len(df["B1"]) - 1)]  
 u[equations[0][0]] = 0  
 print("U" + str(equations[0][0]), "=", 0)  
 while len(equations) != 0:  
 for equation in equations:  
 if u[equation[0]] != sys.maxsize:  
 print("U" + str(equation[0]), "+", "Y" + str(equation[1]), "=", equation[2])  
 new\_y = equation[2] - u[equation[0]]  
 print("Y" + str(equation[1]), "=", new\_y)  
 y[equation[1]] = new\_y  
 equations.remove(equation)  
 else:  
 if y[equation[1]] != sys.maxsize:  
 print("U" + str(equation[0]), "+", "Y" + str(equation[1]), "=", equation[2])  
 new\_u = equation[2] - y[equation[1]]  
 print("U" + str(equation[0]), "=", new\_u)  
 u[equation[0]] = new\_u  
 equations.remove(equation)  
 # Высчитываем относительные оценки  
 for i in range(1, len(df)):  
 for j in range(len(df["B"+str(i)]) - 1):  
 if table.loc[j, "B"+str(i)] == 0:  
 v = df.loc[j, "B" + str(i)] - u[j] - y[i - 1]  
 print("V" + str(j) + str(i - 1), " = ", v)  
 if v < min:  
 min = v  
 res.clear()  
 res = [j, i-1]  
 print("min:", min)  
 res.append(min)  
 return res  
  
  
xl = pd.ExcelFile("Start.xls")  
transport\_task(xl.parse(xl.sheet\_names[0]))

Выводы

Транспортная задача очень эффективна при решении задач перевозки

Заключение

Изученные в ходе выполнения курсовой методы позволяют крайне эффективно решать многокритериальные задачи, задачи линейного программирования и задачи транспортировки грузов

Литература

1. Теория принятия решений. Учебное пособие / А.И.Орлов.- М.: Издательство «Экзамен», 2005. - 656 с.
2. Теория принятия решении и распознавание образов (Панин С.Д.)