Отчёт к лабораторной работе №2

Выполнил: Фроловский А. ИУ7-82

Содержательная постановка

Коммивояжер должен объехать n городов, побывав каждом ровно 1 раз. При этом закончить маршрут он должен в том городе, из которого он начал движение. Стоимость проезда из $i-$ го города в $j-$ й составляет $c_{ij} \geq 0$ единиц (при этом допускается случай $c_{ij} = \infty$ - это означает, что из $i-$ го города нельзя напрямую проехать в $j-$ й город). Необходимо составить маршрут таким образом, чтобы общая сумма переезда была бы минимальной.

Математическая постановка

$$\begin{cases} f(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \to \min \ (1) \\ \sum_{j=1}^{n} x_{ij} = 1, \quad i = \overline{1,n} \ (2) \\ \sum_{i=1}^{n} x_{ij} = 1, \quad j = \overline{1,n} \ (3) \\ (x_{ij}) - \text{полный цикл} \ (4) \\ x_{ij} \in \{0,1\} \ (5) \end{cases},$$

где

$$x_{ij} = \begin{cases} 1, \text{если маршрут коммивояжера включает непосредственный} \\ \text{переезд из } i-\text{го города в } j-\text{й} \\ 0, \text{иначе} \end{cases},$$

$c_{ij} -$ стоимость переезда из $i-$ го города в $j-$ й.

Матрица X задает полный цикл, если существует последовательность назначений индексов $i_1, i_2, \dots, i_n$ такая, что

$$x_{1,i_1} = x_{i_1,i_2} = \ \dots \ = x_{i_{n-1},i_n} = x_{i_n,1} = 1, \quad x_{ij} = 0 - \text{для остальных } i,j.$$

Краткое описание алгоритма

Блок-схема метода ветвей и границ для решения задачи коммивояжера представлена на рис. 1.

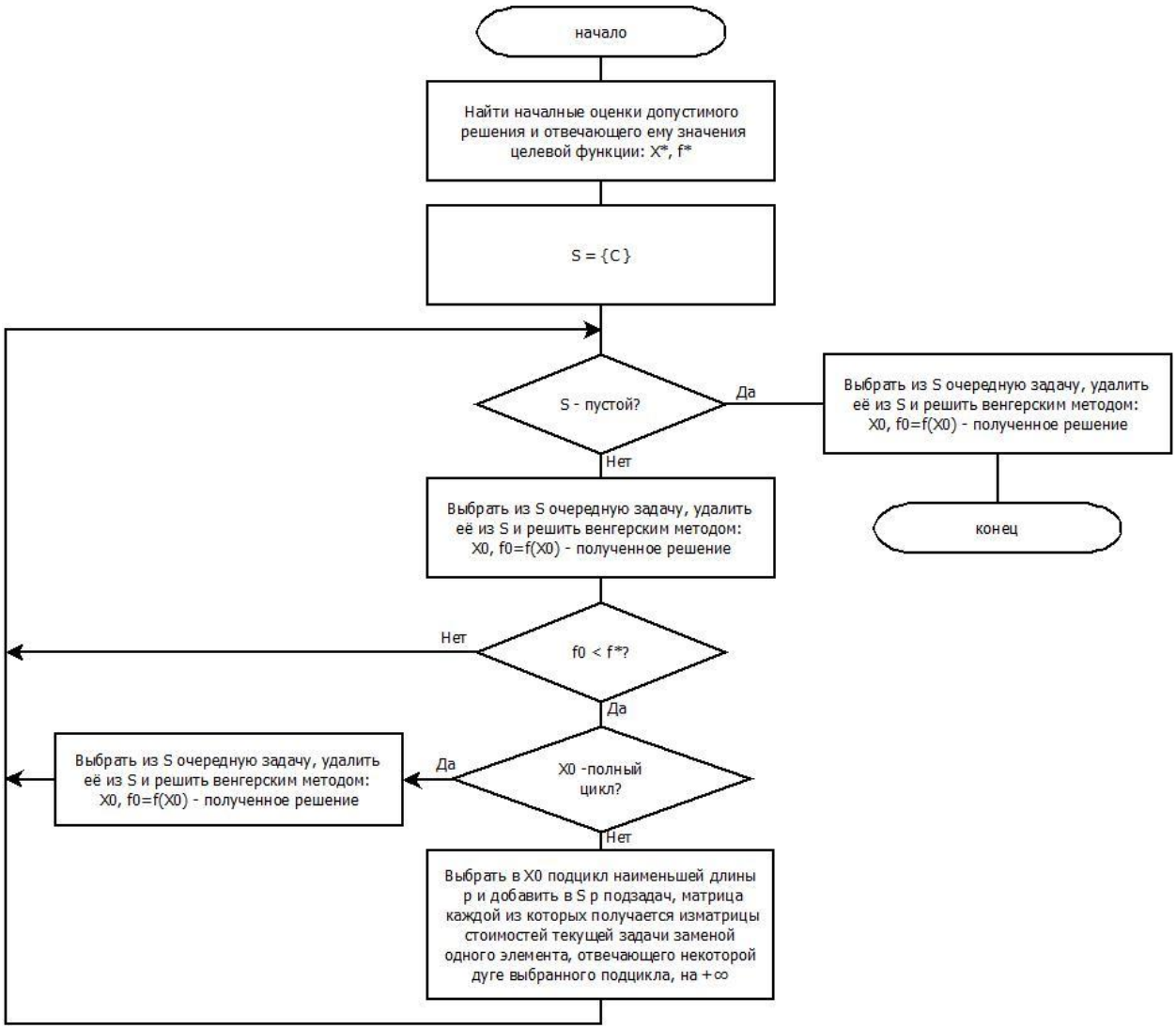

Рис. 1. Блок-схема метода ветвей и границ для решения задачи Коммивояжера.

Текст программы

```
package lab1.pkg1;

import java.util.*;

public class TravelingSalesmanTaskSolution
{
   public TravelingSalesmanTaskSolution(ExtendMatrix costsMatrix) throws Exception
   {
      this.costsMatrix = costsMatrix.copy();
      this.originalCostsMatrix = costsMatrix.copy();
      choicesList = new ArrayList<ExtendMatrix>();
      executeSolution();
   }

   public String getOutput()
   {
      return output;
   }

   public Matrix getAssignmentMatrix()
   {
```

```
    return this.assignmentMatrix;
}

public double getCost()
{
    return getPathCost(this.assignmentMatrix);
}

private void executeSolution() throws Exception
{
    printOutput("\n    Исходная матрица стоимостей:", costsMatrix);
    executeStartSettings();
    printOutput("\n    Первоначальная матрица назначений:", curX);
    printOutput("\n,   Первоначальная стоимость пути:", getPathCost(curX));
    while (!choicesList.isEmpty())
        executeSolutionStep();
    this.assignmentMatrix = curX;
    printOutput("\n    Полученная матрица назначений:", this.assignmentMatrix);
    printOutput("\n    Стоимость пути:", this.getCost());
}

private void executeStartSettings()
{
    int degree = costsMatrix.getDegree();
    curX = getInitialEstimates(degree);
    curF =  getPathCost(curX);
    choicesList = new ArrayList<ExtendMatrix>();
    choicesList.add(costsMatrix.copy());
}

private Matrix getInitialEstimates(int degree)
{
    Matrix res = new Matrix(degree);
    for (int  i = 0; i < degree; i++)
        res = getRowInitialEstimates(res, i);
    return res;
}

private Matrix getRowInitialEstimates(Matrix matrix, int indexOfRow)
{
    int degree = matrix.getDegree();
    for (int j = 0; j < degree; j++)
        if (isInitialEstimatesElement(indexOfRow, j, degree))
            matrix.setVal(indexOfRow, j, 1);
        else
            matrix.setVal(indexOfRow, j, 0);
    return matrix;
}

private boolean isInitialEstimatesElement(int indexOfRow, int indexOfColumn, int degree)
{
    return indexOfColumn == (indexOfRow + 1) % degree;
}

private void executeSolutionStep() throws Exception
{
    ExtendMatrix curMatrix = getCostsMatrixForCurrentTask();
```

```
        printOutput("\n Матрица стоимостей текущей подзадачи:", curMatrix);
        calculateCurrentSolution(curMatrix);
        changeGeneralSolutionOrAddSubtasksIfNecessary(curMatrix);
    }

    private ExtendMatrix getCostsMatrixForCurrentTask()
    {
        ExtendMatrix curMatrix = choicesList.get(0);
        choicesList.remove(0);
        return curMatrix;
    }

    private void calculateCurrentSolution(ExtendMatrix curMatrix)
    {
        HungarianMethod hungarianMethod = new HungarianMethod(curMatrix);
        x0 = hungarianMethod.getAssignmentMatrix();
        output += hungarianMethod.getOutput();
        printOutput("\n    x0:", x0);
        f0 = getPathCost(x0);
        printOutput("\n    f0:", f0);
    }

    private double getPathCost(Matrix x)
    {
        int degree = originalCostsMatrix.getDegree();
        double value = 0;
        for (int  i = 0; i < degree; i++)
            for (int  j = 0; j < degree; j++)
                if (Math.abs(x.getVal(i, j) - 1.0) <= EPS)
                    value += originalCostsMatrix.getVal(i, j);
        return value;
    }

    private void changeGeneralSolutionOrAddSubtasksIfNecessary(ExtendMatrix curMatrix) throws
Exception
    {
        if (f0 < curF)
            if (isCompleteCycle(x0))
                saveNewGeneralSolution();
            else
                addSubtasks(curMatrix);
    }

    private boolean isCompleteCycle(Matrix matrix) throws Exception
    {
        cyclesSet = new CyclesSet(matrix);
        int p = cyclesSet.getMinCycleLength();
        int degree = matrix.getDegree();
        return p == degree;
    }

    private void saveNewGeneralSolution()
    {
        curX = x0.copy();
        printOutput("\n   x*:", curX);
        curF = f0;
        printOutput("\n   f*:", curF);
```

```
    }

    private void addSubtasks(ExtendMatrix matrix) throws Exception
    {
       Cycle minCycle = cyclesSet.getMinCycle();
       int countOfElements = minCycle.getLength();
       for (int i = 0; i < countOfElements; i++)
          addSubtask(i, minCycle, matrix);
       printOutput("\nДобавлено " + countOfElements +
                " подзадачи, общее количество подзадач - " +
                choicesList.size() +"\n");
    }

    private void addSubtask(int indexOfSubtask, Cycle cycle, ExtendMatrix matrix) throws Exception
    {
       int countOfElements = cycle.getLength();
       int indexOfRow = cycle.getElement(indexOfSubtask);
       int indexOfColumn = cycle.getElement((indexOfSubtask + 1)% countOfElements);

       ExtendMatrix res = matrix.copy();
       res.setMarkToElem(indexOfRow, indexOfColumn, (char)8734);
       res.setVal(indexOfRow, indexOfColumn, Double.POSITIVE_INFINITY);
       choicesList.add(res);
    }

    private void printOutput(String prefix, Matrix matrix)
    {
       output += prefix;
       output += matrix.printToString();
    }

    private void printOutput(String prefix, double value)
    {
       output += prefix;
       output += "\n    " + value +"\n";
    }

    private void printOutput(String mes)
    {
       output += mes;
    }


    private CyclesSet cyclesSet;
    private Matrix curX;
    private double curF;
    private Matrix x0;
    private double f0;
    private List<ExtendMatrix> choicesList;
    private Matrix assignmentMatrix;
    private ExtendMatrix costsMatrix;
    private ExtendMatrix originalCostsMatrix;
    private String output = "";
    private final double EPS = 1e-6;
}
```

```
package lab1.pkg1;

import java.util.*;

public class CyclesSet
{
   public CyclesSet(Matrix assignmentMatrix) throws Exception
   {
      this.assignmentMatrix = assignmentMatrix.copy();
      this.cycles = new ArrayList<Cycle>();
      calculateCycles();
      findMinCycle();
   }


   public void calculateCycles() throws Exception
   {
      numberOfLostElements = this.assignmentMatrix.getDegree();
      while (numberOfLostElements != 0)
      {
         Cycle curCycle = getNewCycle();
         cycles.add(curCycle);
      }
   }

   public Cycle getNewCycle() throws Exception
   {
      int startElementIndex = getFreeStartElement();
      Cycle curCycle = new Cycle();
     // curCycle.add(startElementIndex);
     // numberOfLostElements--;;
      int curElementIndex = startElementIndex;
      int lastElementIndex = startElementIndex;

      do{
         curElementIndex = getIndexOfColumnWithOneElement(lastElementIndex);
         curCycle.add(curElementIndex);
         numberOfLostElements--;
         lastElementIndex = curElementIndex;
      }while (startElementIndex != curElementIndex && numberOfLostElements != 0);

      return curCycle;
   }

   public int getIndexOfColumnWithOneElement(int indexOfRow) throws Exception
   {
      int degree = this.assignmentMatrix.getDegree();
      for (int j = 0; j < degree; j++)
         if (this.assignmentMatrix.getVal(indexOfRow, j) == 1)
            return j;
      throw new Exception("Error: not found one element");
   }

   public int getFreeStartElement() throws Exception
   {
      int degree = this.assignmentMatrix.getDegree();
      for (int  i = 0; i < degree; i++)
```

```
            if (isFreeElement(i))
                return i;
        throw new Exception("Error: not found free element");
    }

    public boolean isFreeElement(int elementIndex)
    {
        for (Cycle cycle: cycles)
            if (cycle.isExistElement(elementIndex))
                return false;
        return true;
    }

    public void findMinCycle()
    {
        Object minObj = Collections.min(cycles);
        minCycle = (Cycle) minObj;
    }

    public int getCountOfCycles()
    {
        return cycles.size();
    }

    public Cycle getMinCycle()
    {
        return this.minCycle;
    }

    public int getMinCycleLength()
    {
        return this.minCycle.getLength();
    }

    private int numberOfLostElements;
    private Cycle minCycle;
    private Matrix assignmentMatrix;
    private ArrayList<Cycle> cycles;
}
```

```
package lab1.pkg1;

import java.util.*;

public class Cycle implements Comparable
{
    public Cycle()
    {
        this.sequence = new ArrayList<Integer>();
    }

    public int getLength()
    {
        return this.sequence.size();
    }

    public int getElement(int index) throws Exception
```

```
    {
        try
        {
            return sequence.get(index);
        }
        catch (Exception err)
        {
            throw new Exception("Error: not found elemnt with this index");
        }
    }

    public int getStartElement() throws Exception
    {
        try
        {
            return this.sequence.get(0);
        }
        catch (Exception err)
        {
            throw new Exception("Error: not found first element");
        }
    }

    void add(int el)
    {
        sequence.add(el);
    }

    boolean isExistElement(int element)
    {
        for (Integer el: sequence)
            if (el.equals(element))
                return true;
        return false;
    }


    @Override
    public int compareTo(Object obj)
    {
        if (!(obj instanceof Cycle))
            throw new UnsupportedOperationException("Not supported yet.");
        Cycle cycle = (Cycle) obj;
        int difference = this.getLength() - cycle.getLength();
        return difference;
    }



    private ArrayList<Integer> sequence;
}
```

Результаты расчётов для задач из индивидуальных вариантов

Вариант 6

$$\begin{bmatrix} \infty & 8 & 0 & 8 & 9 \\ 10 & \infty & 7 & 1 & 7 \\ 2 & 11 & \infty & 9 & 10 \\ 8 & 7 & 8 & \infty & 3 \\ 9 & 1 & 10 & 9 & \infty \end{bmatrix}$$

Исходная матрица стоимостей:

∞	8,0	0,0	8,0	9,0
10,0	∞	7,0	1,0	7,0
2,0	11,0	∞	9,0	10,0
8,0	7,0	8,0	∞	3,0
9,0	1,0	10,0	9,0	∞

Первоначальная матрица назначений:

0,0	1,0	0,0	0,0	0,0
0,0	0,0	1,0	0,0	0,0
0,0	0,0	0,0	1,0	0,0
0,0	0,0	0,0	0,0	1,0
1,0	0,0	0,0	0,0	0,0

, Первоначальная стоимость пути:

36.0

Матрица стоимостей текущей подзадачи:

∞	8,0	0,0	8,0	9,0
10,0	∞	7,0	1,0	7,0
2,0	11,0	∞	9,0	10,0
8,0	7,0	8,0	∞	3,0
9,0	1,0	10,0	9,0	∞

∞	8,0	0,0	8,0	9,0
10,0	∞	7,0	1,0	7,0
2,0	11,0	∞	9,0	10,0
8,0	7,0	8,0	∞	3,0
9,0	1,0	10,0	9,0	∞

∞	7,0	0,0*	7,0	6,0
8,0	∞	7,0	0,0*	4,0
0,0*	10,0	∞	8,0	7,0
6,0	6,0	8,0	∞	0,0*
7,0	0,0*	10,0	8,0	∞

x0:

0,0	0,0	1,0	0,0	0,0
0,0	0,0	0,0	1,0	0,0

1,0	0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,0	1,0
0,0	1,0	0,0	0,0	0,0

f0:
7.0

Добавлено 2 подзадачи, общее количество подзадач - 2

Матрица стоимостей текущей подзадачи:

∞	8,0	0,0	8,0	9,0
10,0	∞	7,0	1,0	7,0
∞	11,0	∞	9,0	10,0
8,0	7,0	8,0	∞	3,0
9,0	1,0	10,0	9,0	∞

∞	8,0	0,0	8,0	9,0
10,0	∞	7,0	1,0	7,0
∞	11,0	∞	9,0	10,0
8,0	7,0	8,0	∞	3,0
9,0	1,0	10,0	9,0	∞

∞	7,0	0,0*	7,0	6,0
2,0	∞	7,0	0,0*	4,0
∞	3,0	∞	1,0	0,0*
0,0*	6,0	8,0	∞	0,0
1,0	0,0*	10,0	8,0	∞

x0:

0,0	0,0	1,0	0,0	0,0
0,0	0,0	0,0	1,0	0,0
0,0	0,0	0,0	0,0	1,0
1,0	0,0	0,0	0,0	0,0
0,0	1,0	0,0	0,0	0,0

f0:
20.0

x*:

0,0	0,0	1,0	0,0	0,0
0,0	0,0	0,0	1,0	0,0
0,0	0,0	0,0	0,0	1,0
1,0	0,0	0,0	0,0	0,0
0,0	1,0	0,0	0,0	0,0

f*:
20.0

Матрица стоимостей текущей подзадачи:

∞ 8,0 ∞ 8,0 9,0
10,0 ∞ 7,0 1,0 7,0
2,0 11,0 ∞ 9,0 10,0
8,0 7,0 8,0 ∞ 3,0
9,0 1,0 10,0 9,0 ∞

∞ 8,0 ∞ 8,0 9,0
10,0 ∞ 7,0 1,0 7,0
2,0 11,0 ∞ 9,0 10,0
8,0 7,0 8,0 ∞ 3,0
9,0 1,0 10,0 9,0 ∞

∞ 1,0 ∞ 1,0 0,0*
8,0 ∞ 0,0* 0,0 4,0
0,0* 10,0 ∞ 8,0 7,0
6,0 6,0 1,0 ∞ 0,0
7,0 0,0* 3,0 8,0 ∞

\+ + + +
∞ 1,0 ∞ 1,0 0,0*
8,0 ∞ 0,0* 0,0 4,0
0,0* 10,0 ∞ 8,0 7,0
6,0 6,0 1,0 ∞ 0,0
7,0 0,0* 3,0 8,0 ∞

\+ + + +
∞ 1,0 ∞ 1,0 0,0*
8,0 ∞ 0,0* 0,0' 4,0
0,0* 10,0 ∞ 8,0 7,0
6,0 6,0 1,0 ∞ 0,0
7,0 0,0* 3,0 8,0 ∞

\+ + +
∞ 1,0 ∞ 1,0 0,0*
8,0 ∞ 0,0* 0,0' 4,0 +
0,0* 10,0 ∞ 8,0 7,0
6,0 6,0 1,0 ∞ 0,0
7,0 0,0* 3,0 8,0 ∞

\+ + +
∞ 1,0 ∞ 0,0 0,0*
9,0 ∞ 0,0* 0,0' 5,0 +
0,0* 10,0 ∞ 7,0 7,0
6,0 6,0 0,0 ∞ 0,0
7,0 0,0* 2,0 7,0 ∞

\+ + +
∞ 1,0 ∞ 0,0 0,0*
9,0 ∞ 0,0* 0,0' 5,0 +

0,0*	10,0	∞	7,0	7,0
6,0	6,0	0,0'	∞	0,0
7,0	0,0*	2,0	7,0	∞

∞	1,0	∞	0,0	0,0*
9,0	∞	0,0	0,0*	5,0
0,0*	10,0	∞	7,0	7,0
6,0	6,0	0,0*	∞	0,0
7,0	0,0*	2,0	7,0	∞

x0:

0,0	0,0	0,0	0,0	1,0
0,0	0,0	0,0	1,0	0,0
1,0	0,0	0,0	0,0	0,0
0,0	0,0	1,0	0,0	0,0
0,0	1,0	0,0	0,0	0,0

f0:
21.0

Полученная матрица назначений:

0,0	0,0	1,0	0,0	0,0
0,0	0,0	0,0	1,0	0,0
0,0	0,0	0,0	0,0	1,0
1,0	0,0	0,0	0,0	0,0
0,0	1,0	0,0	0,0	0,0

Стоимость пути:
20.0