МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра СКС

Лабораторна робота №4

з дисципліни

системне програмне забезпечення

на тему:

**«Дослідження принципів проектування та роботи динамічного та статичного планування»**

Виконав студент Перевірив:

Групи КВ-64 \_\_\_\_\_\_\_\_\_\_\_

### Подольський Сергій Валентинович

залікова книжка № КВ6415 \_\_\_\_\_\_\_\_\_\_\_\_\_(бали)

#### VІІ семестр

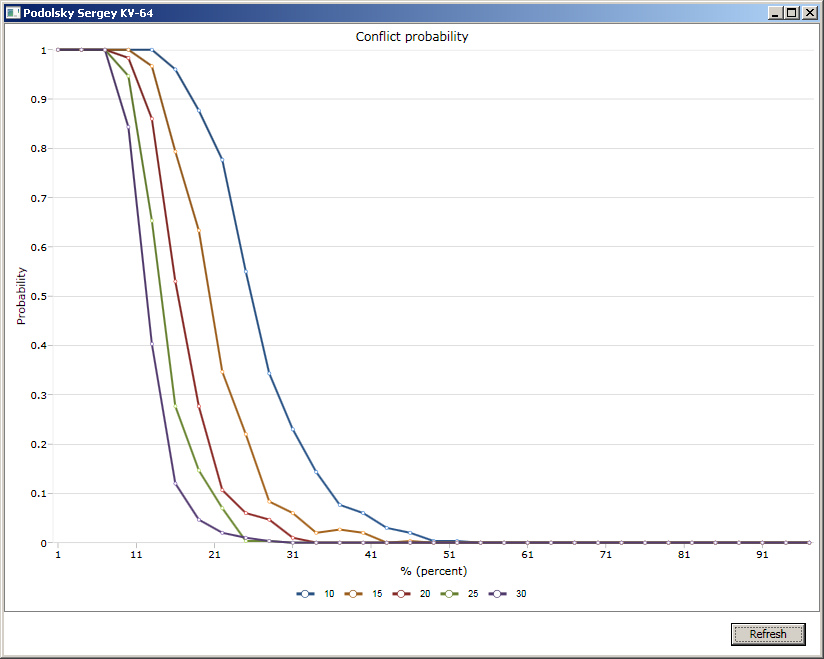
Київ-2009

1. **Постановка задачі**

Варіант № 3:

Визначити вірогідність появи конфліктних призначень при зміні заповненості матриці "1" зв'язності від 1% до 100% для заданої розмірності від 10 до 30 з кроком 5.

1. **Результати роботи програми**

****

1. **Текст програми**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConflictProbabilities

{

static class Matrix

{

/// <summary>

/// Random is used fill matrix

/// </summary>

static Random random = new Random(DateTime.Now.Millisecond);

/// <summary>

/// Swap two rows in the matrix

/// </summary>

/// <typeparam name="T">Type of elements in the matrix</typeparam>

/// <param name="A">Matrix</param>

/// <param name="row1">First row</param>

/// <param name="row2">Second row</param>

static void SwapRows<T>(T[,] A, int row1, int row2)

{

if (row1 == row2) return;

int width = A.GetLength(1);

for (int column = 0; column < width; column++)

{

T temp = A[row1, column];

A[row1, column] = A[row2, column];

A[row2, column] = temp;

}

}

/// <summary>

/// Swap two columns in the matrix

/// </summary>

/// <typeparam name="T">Type of elements in the matrix</typeparam>

/// <param name="A">Matrix</param>

/// <param name="column1">First column</param>

/// <param name="column2">Second column</param>

static void SwapColumns<T>(T[,] A, int column1, int column2)

{

if (column1 == column2) return;

int height = A.GetLength(0);

for (int row = 0; row < height; row++)

{

T temp = A[row, column1];

A[row, column1] = A[row, column2];

A[row, column2] = temp;

}

}

/// <summary>

/// Count of specified elements in specified row

/// </summary>

/// <typeparam name="T">Type of elements in the matrix</typeparam>

/// <param name="A">Matrix</param>

/// <param name="row">Row index</param>

/// <param name="value">Value to search</param>

/// <returns>Count of values</returns>

static int CountInRow<T>(T[,] A, int row, T value)

{

int count = 0;

int width = A.GetLength(1);

for (int column = 0; column < width; column++)

if (A[row, column].Equals(value))

count++;

return count;

}

/// <summary>

/// Count of specified elements in specified column

/// </summary>

/// <typeparam name="T">Type of elements in the matrix</typeparam>

/// <param name="A">Matrix</param>

/// <param name="column">Column index</param>

/// <param name="value">Value to search</param>

/// <returns>Count of values</returns>

static int CountInColumn<T>(T[,] A, int column, T value)

{

int count = 0;

int height = A.GetLength(0);

for (int row = 0; row < height; row++)

if (A[row, column].Equals(value))

count++;

return count;

}

/// <summary>

/// Sort matrix rows due to the count of specified elements in each row

/// </summary>

/// <typeparam name="T">Type of elements in the matrix</typeparam>

/// <param name="A">Matrix</param>

/// <param name="value">Value to encount in each row</param>

static void SortRows<T>(T[,] A, T value)

{

int height = A.GetLength(0);

int[] values = new int[height];

for (int i = 0; i < height; i++)

values[i] = CountInRow<T>(A, i, value);

for (int i = 0; i < height - 1; i++)

{

int jMax = i;

for (int j = i + 1; j < height; j++)

if (values[j] > values[jMax])

jMax = j;

SwapRows<T>(A, jMax, i);

int temp = values[i];

values[i] = values[jMax];

values[jMax] = temp;

}

}

/// <summary>

/// Sort matrix columns due to the count of specified elements in each column

/// </summary>

/// <typeparam name="T">Type of elements in the matrix</typeparam>

/// <param name="A">Matrix</param>

/// <param name="value">Value to encount in each column</param>

static void SortColumns<T>(T[,] A, T value)

{

int width = A.GetLength(1);

int[] values = new int[width];

for (int i = 0; i < width; i++)

values[i] = CountInColumn<T>(A, i, value);

for (int i = 0; i < width - 1; i++)

{

int jMax = i;

for (int j = i + 1; j < width; j++)

if (values[j] > values[jMax])

jMax = j;

SwapColumns<T>(A, jMax, i);

int temp = values[i];

values[i] = values[jMax];

values[jMax] = temp;

}

}

/// <summary>

/// Sort matrix rows and columns due to the count of "zero" cells in each row and column

/// </summary>

/// <param name="A">Matrix</param>

static void Sort(bool[,] A)

{

const bool swapKey = false;

SortRows<bool>(A, swapKey);

SortColumns<bool>(A, swapKey);

}

/// <summary>

/// Create random-filled square matrix of "true" cells

/// </summary>

/// <param name="size">Size of matrix</param>

/// <param name="arcCount">Count of "true" cells</param>

/// <returns>Matrix</returns>

public static bool[,] RandomMatrix(int size, int arcCount)

{

int cellCount = size \* size;

bool[,] matrix = new bool[size, size];

List<int> cellList = new List<int>();

for (int i = 0; i < cellCount; i++)

cellList.Add(i);

for (int i = 0; i < arcCount; i++)

{

int cellIndex = random.Next(cellList.Count);

int cell = cellList[cellIndex];

cellList.RemoveAt(cellIndex);

matrix[cell % size, cell / size] = true;

}

return matrix;

}

/// <summary>

/// Check if submatrix is zero

/// </summary>

/// <param name="A">Parent matrix</param>

/// <param name="width">Submatrix width</param>

/// <param name="height">Submatrix height</param>

/// <returns>Returns true if matrix is zero</returns>

static bool IsZero(bool[,] A, int width, int height)

{

for (int i = 0; i < width; i++)

for (int j = 0; j < height; j++)

if (A[i, j]) return false;

return true;

}

/// <summary>

/// Sort matrix and check either it has zero submatrix

/// </summary>

/// <param name="A">Matrix</param>

/// <returns>Returns true if matrix has at least one conflict</returns>

public static bool HasConflict(bool[,] A)

{

Sort(A);

int size = A.GetLength(0);

for (int width = 1; width < size; width++)

if (IsZero(A, width, size - width + 1)) return true;

return false;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using Visifire.Charts;

using Visifire.Commons;

namespace ConflictProbabilities

{

/// <summary>

/// Interaction logic for Window1.xaml

/// </summary>

public partial class WindowMain : Window

{

/// <summary>

/// Count of experiments per each probability point calcilation

/// </summary>

const int experiments = 300;

/// <summary>

/// Probability increment to go ahead from 0.01 to 1.0

/// </summary>

const double percentStep = 0.03;

/// <summary>

/// Show form

/// </summary>

public WindowMain()

{

InitializeComponent();

// Create Y axis

Axis yAxis = new Axis();

yAxis.AxisMaximum = 1;

yAxis.Title = "Probability";

chart.AxesY.Add(yAxis);

// Create X axis

Axis xAxis = new Axis();

//xAxis.AxisMaximum = 100;

xAxis.Title = "% (percent)";

chart.AxesX.Add(xAxis);

}

/// <summary>

/// Regresh chart

/// </summary>

private void button\_Click(object sender, RoutedEventArgs e)

{

chart.Series.Clear();

// Create series for each matrix size

for (int size = 10; size <= 30; size += 5)

{

// Prepare series

DataSeries series = new DataSeries();

series.RenderAs = RenderAs.Line;

series.ShowInLegend = true;

series.LegendText = size.ToString();

series.LineThickness = 2;

// Get total count of cells in square matrix

int cellCount = size \* size;

// Fill matrix with different percent of "zero" cells (from 1% to 100%)

for (double arcPercent = 0.01; arcPercent <= 1; arcPercent += percentStep)

{

// Calculate statistics of conflicts for all experiments

double conflicts = 0;

for (int experiment = 0; experiment < experiments; experiment++)

if (Matrix.HasConflict(Matrix.RandomMatrix(size, (int)(arcPercent \* cellCount))))

conflicts++;

// Plot point (add to series)

DataPoint point = new DataPoint();

point.XValue = arcPercent \* 100;

point.YValue = conflicts / experiments;

series.DataPoints.Add(point);

}

chart.Series.Add(series);

}

}

}

}