Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Лабораторная работа №3

Исследование дискретно-стохастической СМО

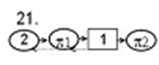
|  |  |  |
| --- | --- | --- |
| Выполнил студент гр. 281011  Радионов М.Г. |  | Проверила  Лашкевич Е.М. |

2015

1. Описание дискретно-стохастической СМО

За основу будет взята дискретно-стохастическая СМО из лабораторной работы №1 со следющим описанием:

|  |  |  |  |
| --- | --- | --- | --- |
| *ρ* | *π*1 | *π*2 | Цель исследования |
| - | 0,4 | VAR | Зависимость Рбл источника от π2,π2=0,2(0,2)1 |



*π*2 будет выбрано со значением 0.8.

Схема содержит источник с блокировкой и фиксированным временем ожидания выдачи заявки (2 такта до выдачи), канал с вероятностями просеивания π1 с блокировкой, накопитель на 1 заявку и канал с вероятностями просеивания π2 (без блокировки).

Граф состояний кодируется четырехкомпонентным вектором *TК1NК2,* где   
 *T*– время до выдачи очередной заявки источником, *T*={2,1,0}

2 – два такта до выдачи заявки

1 – один такт до выдачи заявки (по окончании такта заявка поступит в канал обслуживания)

0 – означает, что источник заблокирован (заявка заблокирована в источнике);

*N* – количество заявок, находящихся в накопителе (длина очереди), *N*={0,1}

0 – заявок в очереди на обслуживание нет

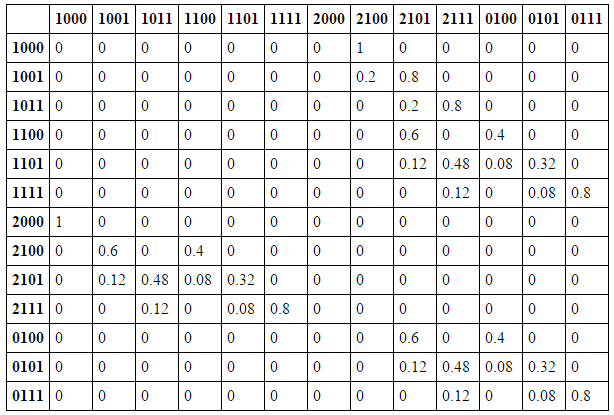
1 – одна заявка в очереди

*К1* и *К2* – состояние каналов обслуживания, *К1*(*К2*)={0,1}

0 – канал свободен

1 – канал занят обслуживанием заявки или заблокирован.

Таблица вероятностей перехода:



2. Реализация цепи Маркова

Для данной таблицы вероятностей переходов определим интервалы для каждого состояния:

|  |  |
| --- | --- |
|  |  |

В качестве примера запустим систему на 10 итераций.



3. Определение искомых параметров

Для расчета вероятности блокировки источника Pбл запустим систему с разным количеством итераций.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество итераций / № запуска | 1 | 2 | 3 | 4 | 5 | Среднее |
| 100 | 0.46 | 0.52 | 0.46 | 0.56 | 0.69 | 0.53 |
| 1000 | 0.54 | 0.59 | 0.63 | 0.58 | 0.57 | 0.58 |
| 10000 | 0.60 | 0.60 | 0.59 | 0.60 | 0.60 | 0.60 |

Pбл = 0.6

Как можно заметить, с увеличением количества итераций возрастает точность определяемых параметров системы, но, с другой стороны, растут требования к вычислительным мощностям и реализации алгоритма.

4. Код программы

'use strict';

// ----------------------------------------------------------------------------

// from lab 1

function getNodes(state) {

var inp = state.split('').map(Number);

var s = inp[0],

c1 = inp[1],

a = inp[2],

c2 = inp[3];

var nodes2 = [];

// console.log('Initial:', s, c1, a ,c2);

if (c2 === 0) {

nodes2.push(newNode(s, c1, a, 0)); // nothing

} else if (c2 === 1) {

nodes2.push(newNode(s, c1, a, 1, undefined, false)); // fail

nodes2.push(newNode(s, c1, a, 0, undefined, true)); // done

} else {

return console.error('Bad c2');

}

nodes2.forEach(function (node) {

if (node.a === 0) {

} else if (node.a === 1) {

if (node.c2 === 0) {

node.c2 = 1;

node.a = 0;

}

} else {

return console.error('Bad a');

}

});

var nodes1 = [];

nodes2.forEach(function (node) {

if (node.c1 === 0) {

nodes1.push(newNode.apply(null, values(node))); // nothing

} else if (node.c1 === 1) {

nodes1.push(newNode(node.s, node.c1, node.a, node.c2, false, node.c2d)); // fail

// done

var nodeDone = newNode(node.s, node.c1, node.a, node.c2, true, node.c2d);

if (nodeDone.a === 0) {

if (nodeDone.c2 === 0) {

nodes1.push(newNode(nodeDone.s, 0, 0, 1, true, nodeDone.c2d));

} else if (nodeDone.c2 === 1) {

nodes1.push(newNode(nodeDone.s, 0, 1, nodeDone.c2, true, nodeDone.c2d));

}

} else if (nodeDone.a === 1) {

nodes1.push(nodeDone);

}

}

});

nodes1.forEach(function (node) {

if (node.s === 0) {

if (node.c1 === 0) {

node.c1 = 1;

node.s = 2;

}

} else if (node.s === 1) {

if (node.c1 === 0) {

node.c1 = 1;

node.s = 2;

} else if (node.c1 === 1) {

node.s = 0;

}

} else if (node.s === 2) {

node.s = 1;

}

});

return arrayfy(nodes1);

}

function newNode(s, c1, a, c2, c1d, c2d) {

return { s: s, c1: c1, a: a, c2: c2, c1d: c1d, c2d: c2d };

}

function arrayfy(nodes) {

return nodes.map(function (node) {

var newNode = {

state: values(node, ['s', 'c1', 'a', 'c2']).join(''),

c1d: node.c1d,

c2d: node.c2d,

};

newNode.formula = getNodeFormula(node);

delete newNode.c1d;

delete newNode.c2d;

return newNode;

});

}

function values(array, keys) {

keys = keys || [];

var result = [];

Object.keys(array).forEach(function (key) {

if (keys.length === 0 || keys.indexOf(key) !== -1) {

result.push(array[key]);

}

});

return result;

}

function size(obj) {

return Object.keys(obj).length;

}

function findWhere(array, condition) {

for (var i = 0; i < array.length; i++) {

var obj = array[i],

key = Object.keys(condition)[0],

value = condition[key];

if (obj[key] === value) {

return obj;

}

}

return undefined;

}

function getNodeFormula(node) {

if (node.c1d === undefined && node.c2d === undefined) {

return '1';

} else if (node.c1d !== undefined && node.c2d === undefined) {

return node.c1d ? '(1 - p1)' : 'p1';

} else if (node.c1d === undefined && node.c2d !== undefined) {

return node.c2d ? '(1 - p2)' : 'p2';

} else if (node.c1d !== undefined && node.c2d !== undefined) {

if (node.c1d && node.c2d) {

return '(1 - p1) \* (1 - p2)';

} else if (node.c1d && !node.c2d) {

return '(1 - p1) \* p2';

} else if (!node.c1d && node.c2d) {

return 'p1 \* (1 - p2)';

} else if (!node.c1d && !node.c2d) {

return 'p1 \* p2';

}

}

}

function buildTree(map, state) {

var nodes = getNodes(state);

if (!map.hasOwnProperty(state)) {

map[state] = [];

nodes.forEach(function (node) {

map[state].push(node);

buildTree(map, node.state);

});

map[state] = collapseNodes(map[state]);

}

}

function collapseNodes(nodes) {

var states = {};

nodes.forEach(function (node) {

states[node.state] = states[node.state] || [];

states[node.state].push(node);

});

if (Object.keys(states).length === nodes.length) {

return nodes;

}

Object.keys(states).forEach(function (state) {

var repnodes = states[state];

if (repnodes.length > 1) {

var formula = repnodes.map(function (repnode) {

return '(' + repnode.formula + ')';

}).join(' + ');

for (var i = 0, l = nodes.length; i < l; i++) {

if (nodes[i].state === state) {

delete nodes[i];

}

}

nodes.push({

state: state,

formula: formula });

}

});

return nodes.filter(function () { return true; });

}

function substitute(formula) {

var f = formula;

f = f.replace(/p1/g, p1);

f = f.replace(/p2/g, p2);

f = (new Function('', 'return ' + f))();

f = f.toFixed(2);

return f;

}

function evaluateTree(tree) {

var map = [];

Object.keys(tree).forEach(function (state) {

var nodes = tree[state];

var row = {

state: state,

nodes: []

};

nodes.forEach(function (node) {

row.nodes.push({

state: node.state,

value: Number(substitute(node.formula))

});

});

map.push(row);

});

return map;

}

function tableizeTree(tree) {

var newTree = {};

var treeStates = Object.keys(tree);

treeStates.forEach(function (outerState) {

var nodes = tree[outerState];

treeStates.forEach(function (innerState) {

newTree[outerState] = newTree[outerState] || {};

newTree[outerState][innerState] = newTree[outerState][innerState] || 0;

var node = findWhere(nodes, { state: innerState });

if (node) {

newTree[outerState][innerState] = Number(node.result);

}

});

});

return newTree;

}

function tableizeMap(map) {

var tableMap = [];

map.forEach(function (outerItem) {

var nodes = outerItem.nodes;

var values = [];

map.forEach(function (innerItem) {

var node = findWhere(nodes, { state: innerItem.state });

if (node) {

values.push({

state: innerItem.state,

value: node.value

});

} else {

values.push({

state: null,

value: 0

});

}

});

tableMap.push({

state: outerItem.state,

nodes: values

});

});

return tableMap;

}

// ----------------------------------------------------------------------------

// from lab 2

function uniformDistribution(sequence, a, b) {

return sequence.map(function (r) {

return a + (b - a) \* r;

});

}

function pickRandom(array) {

return array[Math.floor(array.length \* Math.random())];

}

function generateRandomNumber(from, to) {

from = from || 0;

to = to || 1;

var args = {

x: 7,

a: 15,

c: 3,

n: 32,

i: 100 \* 20

};

args.m = Math.pow(2, args.n);

var seq = [args.x];

for (var i = 0; i < args.i; i++) {

seq.push((args.a \* seq[i] + args.c) % args.m);

}

var normalizedSeq = seq.map(function (number) {

return number / args.m;

});

var distributedSeq = uniformDistribution(normalizedSeq, from, to);

return pickRandom(distributedSeq);

}

// ----------------------------------------------------------------------------

var initialState = '2000';

var p1 = 0.4, p2 = 0.8;

var tree = {};

buildTree(tree, initialState);

var evaluatedMap = evaluateTree(tree);

var tableizedMap = tableizeMap(evaluatedMap);

// -----------------------------------------------------------------------------

var rangeMap = tableizedMap.map(function (tableRow) {

var ranges = [];

var top = 0, bottom = 0;

tableRow.nodes.forEach(function (tableNode) {

if (!tableNode.value) { return; }

bottom = top;

top = Number((top + tableNode.value).toFixed(2));

ranges.push({

state: tableNode.state,

bottom: bottom,

top: top

});

});

return {

state: tableRow.state,

ranges: ranges

};

});

var blockSum = 0;

function move(steps, state) {

steps = steps || 10;

var moves = [],

currentState = state,

ranges,

randomNumber,

move;

for (var i = 0; i < steps; i++) {

if (currentState.charAt(0) == '0') {

blockSum += 1;

}

randomNumber = generateRandomNumber();

ranges = findWhere(rangeMap, { state: currentState }).ranges;

var move = {

fromState: currentState,

randomNumber: randomNumber

};

ranges.forEach(function (range) {

if (range.top < randomNumber || range.bottom > randomNumber) {

return;

}

currentState = range.state;

move.top = range.top;

move.bottom = range.bottom;

move.toState = range.state;

});

moves.push(move);

}

return moves;

}

var iterations = 10000;

var moves = move(iterations, initialState);

console.log('Rbl', (blockSum / iterations).toFixed(2));