import { Injectable } from '@angular/core';

import { LogicService } from './logic.service';

@Injectable()

export class AHPService {

constructor(private logicService: LogicService) { }

calculateDecision(alternatives, criteria,

alternativesComparisons,

criteriaComparisons) {

let alternativesMatrices = [];

let criteriaMatrix = [];

let alternativesPriorities = [];

let criteriaPriorities = [];

let prioritiesMatrix = [];

for(let i=0; i<criteria; i++) {

alternativesMatrices[i] = this.formMatrix(alternativesComparisons[i], alternatives);

alternativesPriorities[i] = this.calculatePriorities(alternativesMatrices[i]);

}

criteriaMatrix = this.formMatrix(criteriaComparisons, criteria);

criteriaPriorities = this.calculatePriorities(criteriaMatrix);

for(let i = 0; i < alternatives; i++){

prioritiesMatrix[i] = [];

for (let j = 0; j < criteria; j++){

prioritiesMatrix[i][j] = alternativesPriorities[j][i]\*criteriaPriorities[j];

}

}

return {criteriaPriorities: criteriaPriorities,

prioritiesMatrix: prioritiesMatrix}

}

getGoalMatrix(prioritiesMatrix) {

let goalMatrix = [];

let rows = prioritiesMatrix.length;

let columns = prioritiesMatrix[0].length;

for(let i = 0; i < rows ; i++) {

goalMatrix[i] = [];

goalMatrix[i][columns] = 0;

for(let j = 0; j < columns ; j++) {

goalMatrix[i][j] = prioritiesMatrix[i][j];

goalMatrix[i][columns] += prioritiesMatrix[i][j];

}

}

return goalMatrix;

}

private formMatrix(combination, lenght) {

let matrix = [];

let rightIndex = 0;

let leftIndex = 0;

for(let i=0; i<lenght; i++) {

matrix[i] = [];

for(let j=i; j<lenght; j++) {

if(i<j){

matrix[i][j] = this.logicService.convertSlider(combination[rightIndex]);

rightIndex++;

}

else {

matrix[i][j] = 1;

}

}

}

for(let j=0; j<lenght; j++) {

for(let i=j+1; i<lenght; i++) {

matrix[i][j] = 1/this.logicService.convertSlider(combination[leftIndex]);

leftIndex++;

}

}

return matrix;

}

private calculatePriorities(matrix){

let priorities = [];

let auxMatrix = [];

let auxVector = [];

let vectorSum = 0;

// Add the columns of the Matrix

for (let j = 0; j < matrix.length; j++) {

auxVector[j] = 0;

for (let i = 0; i < matrix.length; i++) {

auxVector[j] += matrix[i][j];

}

}

// Divide each matrix entry by the corresponding vector entry

for (let i = 0; i < matrix.length; i++) {

auxMatrix[i] = [];

for (let j = 0; j < matrix.length; j++) {

auxMatrix[i][j] = matrix[i][j]/auxVector[j];

}

}

// Add the rows of the aux matrix

auxVector = [];

for (let i = 0; i < auxMatrix.length; i++) {

auxVector[i] = 0;

for (let j = 0; j < auxMatrix.length; j++) {

auxVector[i] += auxMatrix[i][j];

}

}

// Add the entries of the aux vector

for (let i = 0; i < auxVector.length; i++){

vectorSum += auxVector[i];

}

// Divide each aux vector entry by the sum

for (let i = 0; i < auxVector.length; i++){

priorities[i] = auxVector[i]/vectorSum;

}

return priorities;

}

calculateGlobalDecision(matrices, evaluators, alternatives, criteria){

let cube = [];

let goalCube = [];

let globalCriteriaPriorities = [];

let finalVector = [];

cube.push(matrices);

evaluators.forEach(evaluator => {

cube.push(this.calculateDecision(alternatives, criteria,

evaluator.alternativesComparisons, evaluator.criteriaComparisons))

});

for(let i=0; i<criteria; i++) {

globalCriteriaPriorities[i] = [];

for(let j=0; j<cube.length; j++) {

globalCriteriaPriorities[i][j] = cube[j].criteriaPriorities[i]/(cube.length);

}

}

cube.forEach(element => {

goalCube.push(this.getGoalMatrix(element.prioritiesMatrix))

});

for (let i=0; i<alternatives; i++) {

finalVector[i] = 0;

}

goalCube.forEach(matrix => {

for(let i=0; i<matrix.length; i++) {

finalVector[i] += matrix[i][matrix[i].length-1]

}

});

for (let i=0; i<alternatives; i++) {

finalVector[i] /= evaluators.length+1;

}

return {globalCriteriaPriorities: globalCriteriaPriorities,

finalVector: finalVector

}

}

}