**Advanced Computational Techniques**

SSE 635

Dr. MacNeil

Spring 2017

**Project II**

Brent Bitler

Matthew Robison

Table of Contents

[Table of Contents 2](#_Toc475215879)

[Table of Figures 3](#_Toc475215880)

[Table of Tables 3](#_Toc475215881)

[No table of figures entries found. 3](#_Toc475215882)

[Introduction 4](#_Toc475215883)

[Background 4](#_Toc475215884)

[Purpose 5](#_Toc475215885)

[Fuzzy Logic Process 6](#_Toc475215886)

[Overview 6](#_Toc475215887)

[Heating/Cooling Implementation 7](#_Toc475215888)

[Overview 7](#_Toc475215889)

[Testing 8](#_Toc475215890)

[Overview 8](#_Toc475215891)

[Conclusion 9](#_Toc475215892)

[Overview 9](#_Toc475215893)

[Appendix 10](#_Toc475215894)

[BLAH.cs 10](#_Toc475215895)

[Bibliography 11](#_Toc475215896)

[Activity Log - Bitler 12](#_Toc475215897)

[Activity Log - Robison 13](#_Toc475215898)

Table of Figures

[Figure 1. Fuzzy Logic Process Diagram 6](#_Toc475215899)

Table of Tables

**No table of figures entries found.**

Introduction

## Background

SSE 635’s purpose is to educate us on the use of advanced computational techniques and their applications. Project II’s focus is on fuzzy logic, which is rendition of many-valued logic whereupon variables’ truth values may take on any real value between 0 and 1. In essence, it allows a blending of values as a variable may “sort of” be leaning towards true or “sort of” false. An analogy is how tall a basketball player might be. In general, most basketball players are taller by nature.

Some famous players such as Yao Ming or Shaquille O’Neal were both over 7’ tall; however, one of the best three-point shooters in the league right now, Steph Curry, would be considered “short” compared to them at 6’ 3”. It simply does not make sense for there to be a cutoff at 7’, for instance, where someone below such a value is not tall, but anyone over this value is considered tall.

To most, Steph Curry is already tall, but not when stacked up against the previously mentioned players. In this case, Steph would be “slightly tall”, whereas Ming and Shaq would be borderline “giants.” By virtue, they have varying degrees of tallness. If we use Ming and Shaq to help normalize our degree of tallness, they would probably have a value of 0.95 or so, where Curry would end up somewhere near the range of 0.3 since the average height of an NBA player is around 6’7”. If we were to expand this population to include all males in the US, Curry would be above the 0.5 mark as the average height is 5’10”. One can see how fuzzy logic could begin to shape into a useful endeavor to study. It allows for a more meaningful range of measurements to allow a programmer to manipulate for a desired output. Much would be lost in the transition of simply using Boolean logic of 0’s or 1’s when considering this case. Other applications include heating and cooling, autonomous vehicle sensors, or scheduling systems.

## Purpose

For Project II, we were directed to implement an application to demonstrate our mastery of Fuzzy Logic since this is our first course in the realm of computational techniques. Our approach was to implement a simple console application to allow an end user to input a desired room temperature conjunction with the current room temperature. Utilizing the five regions we designate, the program will dictate to what degree of heating or cooling, if at all, needs to be achieved to progress towards the desired temperature.

Fuzzy Logic Process

## Overview

Before explaining how our implementation works, but without regurgitation too much information from the text, we will illustrate a breakdown of the fuzzy rule-based reasoning system as shown in figure 1.

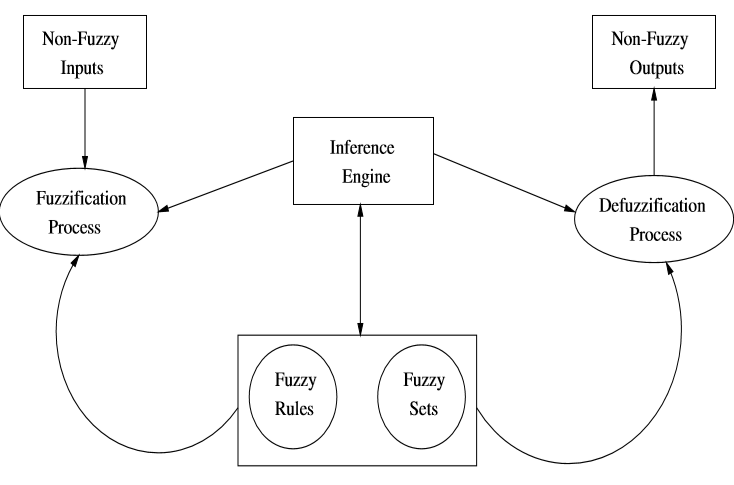


Figure 1. Fuzzy Logic Process Diagram

The fuzzy rules and fuzzy sets make up the knowledge base of the reasoning system. Fuzzification is the process of finding a fuzzy representation of the non-fuzzy inputs through utilization of the membership functions with each fuzzy set in the input space [1]. The inference engine is responsible for producing a fuzzified output for each rule as defined by the logic operators. Finally, defuzzification is used to convert the output of the fuzzy rules into a scalar value [1]. Beyond this, we will assume the reading has a fair grasp on the underlying concepts such as membership functions, unions, intersections, and have read the fuzzy logic sections accompanied in the text. Please refer to these chapters for further clarification before continuing on with the remainder of the paper.

Heating/Cooling Implementation

## Overview

Our console application utilizes fuzzy logic to take an injected desired room temperature and combines this information with a current room temperature to provide the user a mechanism for how much to heat, cool, or not adjust the temperature of the hvac unit to achieve their comfortable living setting. The entry point for our program is illustrated in figures 2 and 3 for the end user to input the current temperature of the room.

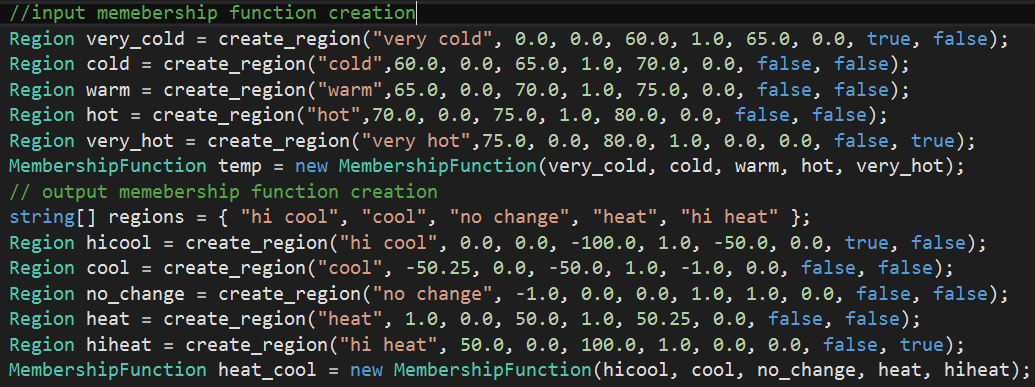


Figure . Input and Output memberships



Figure . Command prompt requiring user input for current temperature

Figure 2 initializes the membership functions of the input temperatures and the output cooling or heating pattern. The input membership function is triangular as depicted in Figure 4.

Figure . Input membership function

Continuing to work through the example, the user has input a current temperature of 67 degrees. This falls under both the cold and warm forms of membership with degrees of membership 0.6 and 0.4, respectively through computation achieved by the EvalMembership method in the MembershipFunction class located in the appendix. This is proven through debugging as seen in Figures 5 and 6.

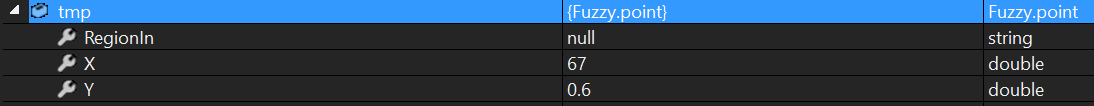


Figure . Degree of membership 0.6 in cold

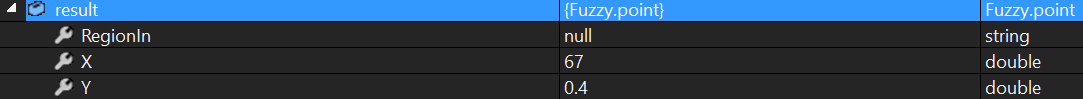


Figure . Degree of membership 0.4 in warm

A dictionary was used as the container with the name and the degree of membership of each region. The degree of membership breakdown in this example is shown in Figure 7.

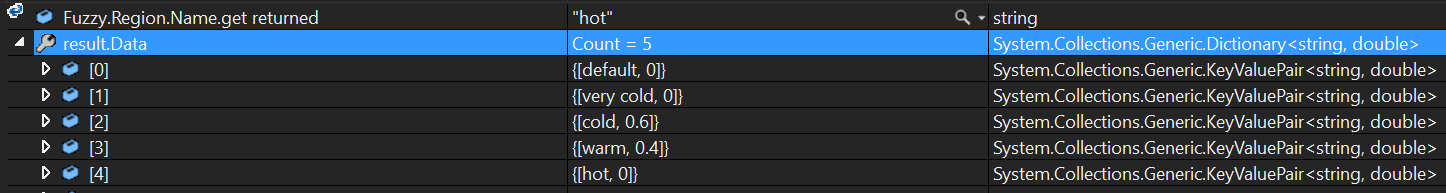


Figure . Degree of membership breakdown for 67 degrees

Figure 7’s calculations are achieved through

Testing

## Overview

Conclusion

## Overview

In conclusion, much was learned about fuzzy logic, its applications, and how to implement the logic to facilitate a real-life system. A more robust application would ultimately take into consideration more thermodynamic properties of a room such as outside temperature, volume of the room, power of the hvac system, or materials of the building. This is beyond the scope of this course, however.

Appendix

## Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Fuzzy

{

class Program

{

static void Main(string[] args)

{

//input memebership function creation

Region very\_cold = create\_region("very cold", 0.0, 0.0, 60.0, 1.0, 65.0, 0.0, true, false);

Region cold = create\_region("cold",60.0, 0.0, 65.0, 1.0, 70.0, 0.0, false, false);

Region warm = create\_region("warm",65.0, 0.0, 70.0, 1.0, 75.0, 0.0, false, false);

Region hot = create\_region("hot",70.0, 0.0, 75.0, 1.0, 80.0, 0.0, false, false);

Region very\_hot = create\_region("very hot",75.0, 0.0, 80.0, 1.0, 0.0, 0.0, false, true);

MembershipFunction temp = new MembershipFunction(very\_cold, cold, warm, hot, very\_hot);

// output memebership function creation

string[] regions = { "hi cool", "cool", "no change", "heat", "hi heat" };

Region hicool = create\_region("hi cool", 0.0, 0.0, -100.0, 1.0, -50.0, 0.0, true, false);

Region cool = create\_region("cool", -50.25, 0.0, -50.0, 1.0, -1.0, 0.0, false, false);

Region no\_change = create\_region("no change", -1.0, 0.0, 0.0, 1.0, 1.0, 0.0, false, false);

Region heat = create\_region("heat", 1.0, 0.0, 50.0, 1.0, 50.25, 0.0, false, false);

Region hiheat = create\_region("hi heat", 50.0, 0.0, 100.0, 1.0, 0.0, 0.0, false, true);

MembershipFunction heat\_cool = new MembershipFunction(hicool, cool, no\_change, heat, hiheat);

// get user input

bool cont = true;

while (cont) {

double current\_temp = 0;

double target\_temp = 0;

bool current\_flag = false;

bool target\_flag = false;

while (!current\_flag)

{

Console.WriteLine("Please Enter a current temperature (ex. 75.0)");

current\_flag = Double.TryParse(Console.ReadLine(), out current\_temp);

}

while (!target\_flag)

{

Console.WriteLine("Please Enter a target temperature (ex. 75.0)");

target\_flag = Double.TryParse(Console.ReadLine(), out target\_temp);

}

FuzzySet current\_fuzz = new FuzzySet();

FuzzySet target\_fuzz = new FuzzySet();

current\_fuzz = temp.EvalMembership(current\_temp, "y");

target\_fuzz = temp.EvalMembership(target\_temp, "y");

FuzzySet output = new FuzzySet();

output = output.BaseRules(target\_fuzz, current\_fuzz);

string result = heat\_cool.defuzzify(output);

Console.WriteLine(result);

Console.WriteLine("would you like to run another execution y/n");

char cont\_char = Convert.ToChar(Console.ReadLine());

if(cont\_char == 'y')

{

cont = true;

}

else

{

cont = false;

}

}

}

static Region create\_region(string name, double leftbound\_x, double leftbound\_y, double peak\_x, double peak\_y, double rightbound\_x, double rightboud\_y, bool leftmost, bool rightmost)

{

if (leftmost && !rightmost)

{

point peak = new point(peak\_x, peak\_y);

point rightbound = new point(rightbound\_x, rightboud\_y);

point leftbound = new point();

Region result = new Region(name,leftbound, rightbound, peak, false, true);

return result;

}

else if (rightmost && !leftmost)

{

point peak = new point(peak\_x, peak\_y);

point rightbound = new point();

point leftbound = new point(leftbound\_x, leftbound\_y);

Region result = new Region(name,leftbound, rightbound, peak, true, false);

return result;

}

else

{

point peak = new point(peak\_x, peak\_y);

point rightbound = new point(rightbound\_x, rightboud\_y);

point leftbound = new point(leftbound\_x, leftbound\_y);

Region result = new Region(name,leftbound, rightbound, peak, false, false);

return result;

}

}

}

}

## Point.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Fuzzy

{

public class point

{

private double x, y;

private string region\_in;

public double X

{

set { x = value; }

get { return x; }

}

public double Y

{

set { y = value; }

get { return y; }

}

public string RegionIn

{

set { region\_in = value; }

get { return region\_in; }

}

public point()

{

x = 0.0;

y = 0.0;

}

public point(double x, double y, string region\_in = "")

{

this.x = x;

this.y = y;

this.region\_in = region\_in;

}

public point liner\_interpolation(point a, point b, double x,string tofind)

{

// (x0,y0) is always the first point passed in;

point result = new point();

if (tofind == "y")

{

double slope = ((b.Y - a.Y) / (b.X - a.X));

double second\_term = x - a.X;

double tmp = a.y + second\_term \* slope;

result.X = x;

result.Y = tmp;

return result;

}

else

{

double slope = ((b.X - a.X) / (b.Y - a.Y));

double second\_term = x - a.Y;

double tmp = a.X + second\_term \* slope;

result.X = tmp;

result.Y = x;

return result;

}

}

}

}

## Region.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Fuzzy

{

class Region

{

string name;

bool leftmost;

bool rightmost;

private point leftbound;

private point rightbound;

private point peak;

public point Leftbound

{

get { return leftbound; }

set { leftbound = value;}

}

public point Rightbound

{

get { return rightbound; }

set { rightbound = value; }

}

public point Peak

{

get { return peak; }

set { peak = value; }

}

public bool Leftmost

{

get { return leftmost; }

set { leftmost = value; }

}

public bool Rightmost

{

get { return rightmost; }

set { rightmost = value; }

}

public string Name

{

get { return name; }

set { name = value; }

}

public Region()

{

point leftbound = new point();

this.leftbound = leftbound;

point rightbound = new point();

this.rightbound = rightbound;

point peak = new point();

this.peak = peak;

}

public Region(string name, point leftbound, point rightbound , point peak, bool rightmost = false, bool leftmost = false)

{

this.name = name;

this.leftbound = leftbound;

this.rightbound = rightbound;

this.peak = peak;

this.leftmost = leftmost;

this.rightmost = rightmost;

}

}

}

## MembershipFunction.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Fuzzy

{

class MembershipFunction

{

Region[] regions;

public Region[] Regions

{

get { return regions; }

set { regions = value; }

}

public MembershipFunction()

{

}

#region Overloads of MembershipFunction Constructors

public MembershipFunction(Region one)

{

regions = new Region[1];

regions[0] = one;

}

public MembershipFunction(Region one, Region two)

{

regions = new Region[2];

regions[0] = one;

regions[1] = two;

}

public MembershipFunction(Region one, Region two, Region three)

{

regions = new Region[3];

regions[0] = one;

regions[1] = two;

regions[2] = three;

}

public MembershipFunction(Region one, Region two, Region three, Region four)

{

regions = new Region[4];

regions[0] = one;

regions[1] = two;

regions[2] = three;

regions[3] = four;

}

#endregion

public MembershipFunction(Region one, Region two, Region three, Region four, Region five)

{

regions = new Region[5];

this.regions[0] = one;

this.regions[1] = two;

this.regions[2] = three;

this.regions[3] = four;

this.regions[4] = five;

}

public FuzzySet EvalMembership(double input, string tofind)

{

FuzzySet result = new FuzzySet();

for (int i = 0; i < regions.Length; i++)

{

if(regions[i].Leftmost == true && regions[i].Rightmost == false)

{

// is the leftmost region

if( input <= regions[i].Rightbound.X && input > regions[i].Peak.X)

{

point tmp = new point();

tmp = tmp.liner\_interpolation(regions[i].Peak, regions[i].Rightbound, input, tofind);

result.Data.Add(regions[i].Name, tmp.Y);

}

else if(input <= regions[i].Peak.X)

{

result.Data.Add(regions[i].Name, 1.0);

}

else

{

result.Data.Add(regions[i].Name, 0.0);

}

}

else if(regions[i].Rightmost == true && regions[i].Leftmost == false)

{

// is the rightmost region

if(input >= regions[i].Leftbound.X && input < regions[i].Peak.X)

{

point tmp = new point();

tmp = tmp.liner\_interpolation(regions[i].Peak, regions[i].Leftbound, input, tofind);

result.Data.Add(regions[i].Name, tmp.Y);

}

else if(input >= regions[i].Peak.X)

{

result.Data.Add(regions[i].Name, 1.0);

}

else

{

result.Data.Add(regions[i].Name, 0.0);

}

}

else

{

// middle region

if(input >= regions[i].Leftbound.X && input < regions[i].Peak.X)

{

point tmp = new point();

tmp = tmp.liner\_interpolation(regions[i].Leftbound, regions[i].Peak, input, tofind);

result.Data.Add(regions[i].Name, tmp.Y);

}

else if(input >= regions[i].Peak.X && input <= regions[i].Rightbound.X)

{

point tmp = new point();

tmp = tmp.liner\_interpolation(regions[i].Peak, regions[i].Rightbound, input, tofind);

result.Data.Add(regions[i].Name, tmp.Y);

}

else

{

result.Data.Add(regions[i].Name, 0.0);

}

}

}

return result;

}

public string defuzzify(FuzzySet data, string type = "max")

{

type.ToLower();

if (type == "max")

{

double max = 0.0;

string region = string.Empty;

foreach (string key in data.Data.Keys)

{

if (data.Data[key] != 0.0)

{

if (data.Data[key] > max)

{

max = data.Data[key];

region = key;

}

}

}

string[] regions = { "hi cool", "cool", "no change", "heat", "hi heat" };

point value = new point();

point value1 = new Fuzzy.point();

if (region == regions[0])

{

value = value.liner\_interpolation(this.Regions[0].Peak, this.Regions[0].Leftbound, max, "x");

return "Output is: " + Math.Abs(value.X) + "% cooling";

}

else if(region == regions[1])

{

value = value.liner\_interpolation(this.Regions[1].Peak, this.Regions[1].Leftbound, max, "x");

value1 = value1.liner\_interpolation(this.Regions[1].Peak, this.Regions[1].Rightbound, max, "x");

double output = (value.X + value1.X) / 2;

return "Output is: " + Math.Abs(output) + "% cooling";

}

else if (region == regions[2])

{

value = value.liner\_interpolation(this.Regions[2].Peak, this.Regions[2].Leftbound, max, "x");

value1 = value1.liner\_interpolation(this.Regions[2].Peak, this.Regions[2].Rightbound, max, "x");

double output = (value.X + value1.X) / 2;

return "Output is: " + Math.Abs(output) + "% sytstem is off";

}

else if (region == regions[3])

{

value = value.liner\_interpolation(this.Regions[3].Peak, this.Regions[3].Leftbound, max, "x");

value1 = value1.liner\_interpolation(this.Regions[3].Peak, this.Regions[3].Rightbound, max, "x");

double output = (value.X + value1.X) / 2;

return "Output is: " + Math.Abs(output) + "% heating";

}

else if (region == regions[4])

{

value = value.liner\_interpolation(this.Regions[4].Peak, this.Regions[4].Rightbound, max, "x");

return "Output is: " + Math.Abs(value.X) + "% heating";

}

else

{

return string.Empty;

}

}

else if (type == "gravity")

{

// logic for doing center of gravity method

return string.Empty;

}

else

{

// logic for doing center of area

return string.Empty;

}

}

}

}

## FuzzySet.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Fuzzy

{

class FuzzySet

{

Dictionary<string, double> data;

string name;

public Dictionary<string,double> Data

{

get { return data; }

set { data = value; }

}

public string Name

{

get { return name; }

set { name = value; }

}

public FuzzySet()

{

// create a brand new object with no data

this.data = new Dictionary<string, double>();

data.Add("default", 0.0);

name = "";

string[] region\_name = new string[0];

double[] region\_dom = new double[0];

}

public FuzzySet(string[] region, double[] dom, string name)

{

// create object with data;

this.data = new Dictionary<string, double>(50);

this.name = name;

if (dom.Length == region.Length)

{

for (int i = 0; i < region.Length; i++)

{

data.Add(region[i], dom[i]);

}

}

}

public FuzzySet BaseRules(FuzzySet target\_tmp, FuzzySet current\_tmp)

{

List<double> HiCool = new List<double>();

List<double> Cool = new List<double>();

List<double> NoChange = new List<double>();

List<double> Heat = new List<double>();

List<double> HiHeat = new List<double>();

HiCool.Add(0.0);

Cool.Add(0.0);

NoChange.Add(0.0);

Heat.Add(0.0);

HiHeat.Add(0.0);

////////////+---------------------------------------------------------------------+

////////////| target Temp | Knowledge Base |

////////////+-----------------------+---------+---------+----------+--------------+

////////////| Current temp | Very Cold | Cold | Warm | Hot | Very Hot |

////////////+---------------------------------------------------------------------+

////////////| Very Cold | No Change | heat | hi-heat | hi-heat | hi-heat |

////////////+---------------------------------------------------------------------+

////////////| Cold | cool | no change | heat | hi-heat | hi-heat |

////////////+---------------------------------------------------------------------+

////////////| Warm | hi-cool | cool | No Change | heat | hi-heat |

////////////+---------------------------------------------------------------------+

////////////| Hot | hi-cool | hi-cool | cool | No Change | heat |

////////////+---------------------------------------------------------------------+

////////////| Very Hot | hi-cool | hi-cool | hi-cool | cool | No Change |

////////////+------------+----------+---------+---------+----------+--------------+

// no\_change senerios

if (target\_tmp.Data["very cold"] != 0.0 && current\_tmp.Data["very cold"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["very cold"], current\_tmp.Data["very cold"]);

NoChange.Add(tmp);

}

if (target\_tmp.Data["cold"] != 0.0 && current\_tmp.Data["cold"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["cold"], current\_tmp.Data["cold"]);

NoChange.Add(tmp);

}

if (target\_tmp.Data["warm"] != 0.0 && current\_tmp.Data["warm"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["warm"], current\_tmp.Data["warm"]);

NoChange.Add(tmp);

}

if (target\_tmp.Data["hot"] != 0.0 && current\_tmp.Data["hot"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["hot"], current\_tmp.Data["hot"]);

NoChange.Add(tmp);

}

if (target\_tmp.Data["very hot"] != 0.0 && current\_tmp.Data["very hot"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["very hot"], current\_tmp.Data["very hot"]);

NoChange.Add(tmp);

}

if (target\_tmp.Data["very cold"] != 0.0 && current\_tmp.Data["cold"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["very cold"], current\_tmp.Data["cold"]);

Cool.Add(tmp);

}

if (target\_tmp.Data["very cold"] != 0.0 && (current\_tmp.Data["warm"] != 0.0 || current\_tmp.Data["hot"] != 0.0 || current\_tmp.Data["very hot"] != 0.0))

{

double tmp = Math.Min(target\_tmp.Data["very cold"], current\_tmp.Data["warm"]);

double tmp1 = Math.Min(target\_tmp.Data["very cold"], current\_tmp.Data["hot"]);

double tmp2 = Math.Min(target\_tmp.Data["very cold"], current\_tmp.Data["very hot"]);

double value = Math.Max(tmp1, tmp);

double value2 = Math.Max(value, tmp2);

HiCool.Add(value2);

}

if (target\_tmp.Data["cold"] != 0.0 && current\_tmp.Data["very cold"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["cold"], current\_tmp.Data["very cold"]);

Heat.Add(tmp);

}

if (target\_tmp.Data["cold"] != 0.0 && current\_tmp.Data["warm"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["cold"], current\_tmp.Data["warm"]);

Cool.Add(tmp);

}

if (target\_tmp.Data["cold"] != 0.0 && current\_tmp.Data["hot"] != 0.0 || current\_tmp.Data["very hot"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["cold"], current\_tmp.Data["hot"]);

double tmp2 = Math.Min(target\_tmp.Data["cold"], current\_tmp.Data["very hot"]);

double value = Math.Max(tmp, tmp2);

HiCool.Add(value);

}

if(target\_tmp.Data["warm"] != 0.0 && current\_tmp.Data["very cold"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["warm"], current\_tmp.Data["very cold"]);

HiHeat.Add(tmp);

}

if(target\_tmp.Data["warm"] != 0.0 && current\_tmp.Data["cold"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["warm"], current\_tmp.Data["cold"]);

Heat.Add(tmp);

}

if(target\_tmp.Data["warm"] != 0.0 && current\_tmp.Data["hot"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["warm"], current\_tmp.Data["hot"]);

Cool.Add(tmp);

}

if(target\_tmp.Data["warm"] != 0.0 && current\_tmp.Data["very hot"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["warm"], current\_tmp.Data["very hot"]);

HiCool.Add(tmp);

}

if(target\_tmp.Data["hot"] != 0.0 && (current\_tmp.Data["very cold"] != 0.0 || current\_tmp.Data["cold"] != 0.0))

{

double tmp = Math.Min(target\_tmp.Data["hot"], current\_tmp.Data["very cold"]);

double tmp2 = Math.Min(target\_tmp.Data["hot"], current\_tmp.Data["cold"]);

double value = Math.Max(tmp, tmp2);

HiHeat.Add(value);

}

if(target\_tmp.Data["hot"] != 0.0 && current\_tmp.Data["warm"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["hot"], current\_tmp.Data["warm"]);

Heat.Add(tmp);

}

if (target\_tmp.Data["hot"] != 0.0 && current\_tmp.Data["very hot"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["hot"], current\_tmp.Data["very hot"]);

Cool.Add(tmp);

}

if(target\_tmp.Data["very hot"] != 0.0 && (current\_tmp.Data["very cold"] != 0.0 || current\_tmp.Data["cold"] != 0.0 || current\_tmp.Data["warm"] != 0.0))

{

double tmp = Math.Min(target\_tmp.Data["very hot"], current\_tmp.Data["very cold"]);

double tmp2 = Math.Min(target\_tmp.Data["very hot"], current\_tmp.Data["cold"]);

double tmp3 = Math.Min(target\_tmp.Data["very hot"], current\_tmp.Data["warm"]);

double value = Math.Max(tmp, tmp2);

double value2 = Math.Max(tmp3, value);

HiHeat.Add(value2);

}

if(target\_tmp.Data["very hot"] != 0.0 && current\_tmp.Data["hot"] != 0.0)

{

double tmp = Math.Min(target\_tmp.Data["very hot"], current\_tmp.Data["hot"]);

Heat.Add(tmp);

}

double max\_hicool = 0.0;

string[] regions = { "hi cool", "cool", "no change", "heat", "hi heat" };

double[] dom = new double[5];

foreach (double item in HiCool)

{

max\_hicool = Math.Max(max\_hicool, item);

}

dom[0] = max\_hicool;

double max\_cool = 0.0;

foreach (double item in Cool)

{

max\_cool = Math.Max(max\_cool, item);

}

dom[1] = max\_cool;

double no\_change = 0.0;

foreach (double item in NoChange)

{

no\_change = Math.Max(no\_change, item);

}

dom[2] = no\_change;

double max\_heat = 0.0;

foreach (double item in Heat)

{

max\_heat = Math.Max(max\_heat, item);

}

dom[3] = max\_heat;

double max\_hiheat = 0.0;

foreach (double item in HiHeat)

{

max\_hiheat = Math.Max(max\_hiheat, item);

}

dom[4] = max\_hiheat;

FuzzySet result = new FuzzySet(regions, dom, "output");

return result;

}

}

}

Bibliography

[1] Engelbrecht, Andries P. Computational Intelligence: An Introduction, (2nd ed.). Wiley, 2007.

Activity Log - Bitler

|  |  |  |
| --- | --- | --- |
| Date | Time (mins) | Description |
| *1/9/17* | *60* | *Looked through Naïve GA* |
| *1/10/17* | *60* | *Looked through and worked Naïve GA* |
| *1/11/17* |  |  |
| *1/12/17* | *120* | *Research GA’s online for topic ideas* |
| *1/13/17* | *180* | *Read Ch 9 - GA* |
| *1/14/17* | *90* | *Read Ch 9 - GA* |
| *1/15/17* |  |  |
| *1/16/17* | *120* | *Begin constructing class structure of string unscramble program* |
| *1/17/17* | *120* | *Begin chromosome class* |
| *1/18/17* | *120* | *Continue working chromosome class, begin developing population class* |
| *1/19/17* |  |  |
| *1/20/17* |  |  |
| *1/21/17* | *60* | *Finish chromosome class* |
| *1/22/17* | *240* | *Continue working population class* |
| *1/23/17* | *120* | *Connector class* |
| *1/24/17* | *30* | *Come up with initial variable settings* |
| *1/25/17* |  |  |
| *1/26/17* | *240* | *Begin generating report* |
| *1/27/17* | *180* | *Put together test data* |
| *1/28/17* | *360* | *Conclude paper, graphs, etc* |
| *1/29/17* | *180* | *Final review of paper* |
| *1/30/17* | *15* | *Submit paper* |
| 1/31/17 |  |  |
| 2/1/17 | 90 | Begin reading Ch 20 – Fuzzy sets |
| 2/2/17 | 120 | Continue reading Ch 20 – Fuzzy sets |
| 2/3/17 | 120 | Begin Ch 21 – Fuzzy Logic and Reasoning |
| 2/4/17 |  |  |
| 2/5/17 | 120 | Continue Ch 21 – Fuzzy Logic and Reasoning |
| 2/6/17 | 90 | Read Ch 22, 23 – Fuzzy controllers and Rough sets |
| 2/7/17 | 120 | Watch tutorials on fuzzy logic |
| 2/8/17 | 120 | Develop class structure for Heating/Cooling app |
| 2/9/17 | 120 | Work on FuzzySet, MembershipFunction, and Region classes |
| 2/10/17 | 180 | Continue class development |
| 2/11/17 | 120 | Work on Point class |
| 2/12/17 |  |  |
| 2/13/17 | 300 | Begin generating report |
| 2/14/17 |  |  |
| 2/15/17 | 360 | Continue report |
| 2/16/17 | 240 | Continue report |
| 2/17/17 | 240 | Conclude paper, graphs, etc |
| 2/18/17 | 45 | Final review of paper |
| 2/19/17 |  |  |
| 2/20/17 | 15 | Submit paper |
| *Project 1* | *2295* |  |
| Project 2 | 2400 |  |
| TOTAL | 4695 |  |

Activity Log - Robison