Erasmus Mundus

Facultad de Informática

Universidad Politécnica de Madrid

European Master In Computational Logic

Practical Project

Implementation of An Atomized Methodology for Evaluating Credibility Values Concerning Similarity Relations in Fuzzy Logic

Submitted by: Sinan Eğilmez

Supervisor: Susana Muñoz Hernández
Co-supervisor: Victor Pablos Ceruelo

October 2012

Contents

Chapter 1

Practical Application

In [1] we introduce our own methodology for evaluating the similarity degree of fuzzy predicates. The method lacked a feature for realizing an error tolerance when working on knowledge bases with incomplete information. For that reason, three approaches are proposed in [1] . The most elegant of these approaches, namely the *Hybrid Approach* inherits the best features of the preceding ones, while avoiding the shortcomings via merging the two by attaining them some weights. Previously we displayed how these weights could be decided in an intuitionistic way, and hinted the possibility of an atomized procedure. In this chapter we show our proposal for atomizing the process with a real-world example.

1.1 The Problem

In the introduction chapter of [1], we mention about how today's search engines have limited querying capabilities. We give the example of the query *red car*, and say that an ideal frame work would return us the results of the query *orange car* in addition to the original one, with lower credibility values. In the light of this, as our real-world practical example, we inspect the similarity relations between the colors. We may observe colors as a domain of interest in many distinct fields, however one particular interesting example is the set of canonical colors that the web browsers use, namely the *X11 colors* [4]. The system makes use of the *RGB* framework, where colors are specified as triplets. Every color is depicted by three values which represent the *Red*, *Green* and *Blue* amount that the color consists of. The complete *X11* table is shown in the following page.

HTML name	Hex code R G B	Decimal code R G B
Red colors		
IndianRed	CD 5C 5C	205 92 92
LightCoral	F0 80 80	240 128 128
Salmon	FA 80 72	250 128 114
DarkSalmon	E9 96 7A	233 150 122
LightSalmon	FF A0 7A	255 160 122
Red	FF 00 00	255 0 0
Crimson	DC 14 3C	220 20 60
FireBrick	B2 22 22	178 34 34
DarkRed	8B 00 00	139 0 0
Pink colors		
Pink	FF C0 CB	255 192 203
LightPink	FF B6 C1	255 182 193
HotPink	FF 69 B4	255 105 180
DeepPink	FF 14 93	255 20 147
MediumVioletRed	C7 15 85	199 21 133
PaleVioletRed	DB 70 93	219 112 147
Orange colors		
LightSalmon	FF A0 7A	255 160 122
Coral	FF 7F 50	255 127 80
Tomato	FF 63 47	255 99 71
OrangeRed	FF 45 00	255 69 0
DarkOrange	FF 8C 00	255 140 0
Orange	FF A5 00	255 165 0
Yellow colors		
Gold	FF D7 00	255 215 0
Yellow	FF FF 00	255 255 0
LightYellow	FF FF E0	255 255 224
LemonChiffon	FF FA CD	255 250 205
LightGoldenrodYellow	FA FA D2	250 250 210
PapayaWhip	FF EF D5	255 239 213
Moccasin	FF E4 B5	255 228 181
PeachPuff	FF DA B9	255 218 185
PaleGoldenrod	EE E8 AA	238 232 170
Khaki	F0 E6 8C	240 230 140
DarkKhaki	BD B7 6B	189 183 107
Purple colors		
Lavender	E6 E6 FA	230 230 250
Thistle	D8 BF D8	216 191 216
Plum	DD A0 DD	221 160 221
Violet	EE 82 EE	238 130 238
Orchid	DA 70 D6	218 112 214
Fuchsia	FF 00 FF	255 0 255
Magenta	FF 00 FF	255 0 255
MediumOrchid	BA 55 D3	186 85 211
MediumPurple	93 70 DB	147 112 219
BlueViolet	8A 2B E2	138 43 226
DarkViolet	94 00 D3	148 0 211
DarkOrchid	99 32 CC	153 50 204
DarkMagenta	8B 00 8B	139 0 139
Purple	80 00 80	128 0 128
Indigo	4B 00 82	75 0 130
DarkSlateBlue	48 3D 8B	72 61 139
SlateBlue	6A 5A CD	106 90 205
MediumSlateBlue	7B 68 EE	123 104 238

HTML name	Hex code R G B	Decimal code R G B
Green colors		
GreenYellow	AD FF 2F	173 255 47
Chartreuse	7F FF 00	127 255 0
LawnGreen	7C FC 00	124 252 0
Lime	00 FF 00	0 255 0
LimeGreen	32 CD 32	50 205 50
PaleGreen	98 FB 98	152 251 152
LightGreen	90 EE 90	144 238 144
MediumSpringGreen	00 FA 9A	0 250 154
SpringGreen	00 FF 7F	0 255 127
MediumSeaGreen	3C B3 71	60 179 113
SeaGreen	2E 8B 57	46 139 87
ForestGreen	22 8B 22	34 139 34
Green	00 80 00	0 128 0
DarkGreen	00 64 00	0 100 0
YellowGreen	9A CD 32	154 205 50
OliveDrab	6B 8E 23	107 142 35
Olive	80 80 00	128 128 0
DarkOliveGreen	55 6B 2F	85 107 47
MediumAquamarine	66 CD AA	102 205 170
DarkSeaGreen	8F BC 8F	143 188 143
LightSeaGreen	20 B2 AA	32 178 170
DarkCyan	00 8B 8B	0 139 139
Teal	00 80 80	0 128 128
Blue/Cyan colors		
Aqua	00 FF FF	0 255 255
Cyan	00 FF FF	0 255 255
LightCyan	E0 FF FF	224 255 255
PaleTurquoise	AF EE EE	175 238 238
Aquamarine	7F FF D4	127 255 212
Turquoise	40 E0 D0	64 224 208
MediumTurquoise	48 D1 CC	72 209 204
DarkTurquoise	00 CE D1	0 206 209
CadetBlue	5F 9E A0	95 158 160
SteelBlue	46 82 B4	70 130 180
LightSteelBlue	B0 C4 DE	176 196 222
PowderBlue	B0 E0 E6	176 224 230
LightBlue	AD D8 E6	173 216 230
SkyBlue	87 CE EB	135 206 235
LightSkyBlue	87 CE FA	135 206 250
DeepSkyBlue	00 BF FF	0 191 255
DodgerBlue	1E 90 FF	30 144 255
CornflowerBlue	64 95 ED	100 149 237
RoyalBlue	41 69 E1	65 105 225
Blue	00 00 FF	0 0 255
MediumBlue	00 00 CD	0 0 205
DarkBlue	00 00 8B	0 0 139
Navy	00 00 80	0 0 128
MidnightBlue	19 19 70	25 25 112

HTML name	Hex code R G B	Decimal code R G B
Brown colors		
Cornsilk	FF F8 DC	255 248 220
BlanchedAlmond	FF EB CD	255 235 205
Bisque	FF E4 C4	255 228 196
NavajoWhite	FF DE AD	255 222 173
Wheat	F5 DE B3	245 222 179
BurlyWood	DE B8 87	222 184 135
Tan	D2 B4 8C	210 180 140
RosyBrown	BC 8F 8F	188 143 143
SandyBrown	F4 A4 60	244 164 96
Goldenrod	DA A5 20	218 165 32
DarkGoldenrod	B8 86 0B	184 134 11
Peru	CD 85 3F	205 133 63
Chocolate	D2 69 1E	210 105 30
SaddleBrown	8B 45 13	139 69 19
Sienna	A0 52 2D	160 82 45
Brown	A5 2A 2A	165 42 42
Maroon	80 00 00	128 0 0
White colors		
White	FF FF FF	255 255 255
Snow	FF FA FA	255 250 250
Honeydew	F0 FF F0	240 255 240
MintCream	F5 FF FA	245 255 250
Azure	F0 FF FF	240 255 255
AliceBlue	F0 F8 FF	240 248 255
GhostWhite	F8 F8 FF	248 248 255
WhiteSmoke	F5 F5 F5	245 245 245
Seashell	FF F5 EE	255 245 238
Beige	F5 F5 DC	245 245 220
OldLace	FD F5 E6	253 245 230
FloralWhite	FF FA F0	255 250 240
Ivory	FF FF F0	255 255 240
AntiqueWhite	FA EB D7	250 235 215
Linen	FA F0 E6	250 240 230
LavenderBlush	FF F0 F5	255 240 245
MistyRose	FF E4 E1	255 228 225
Gray colors		
Gainsboro	DC DC DC	220 220 220
LightGrey	D3 D3 D3	211 211 211
Silver	C0 C0 C0	192 192 192
DarkGray	A9 A9 A9	169 169 169
Gray	80 80 80	128 128 128
DimGray	69 69 69	105 105 105
LightSlateGray	77 88 99	119 136 153
SlateGray	70 80 90	112 128 144
DarkSlateGray	2F 4F 4F	47 79 79
Black	00 00 00	0 0 0

1.1.1 Preliminaries

Before starting to proceed with the problem itself, we ought to mention about some problem specific concepts.

1.1.1.1 Real World Similarity

As stated in section 1.1 in the *RGB* scheme, all colors are represented by triplets of *Red*, *Green* and *Blue* values. With this in mind, when observing the similarity value between two colors, directly from the real-word data, we may consider them as if they were placed in *3D space*. Thus each of the *RGB* values acts as a coordinate for the corresponding color, and we may utilize a geometric distance formulation for this problem. In this scenario we adopt the *Euclidean distance*, which is indeed one of the common definitions in the field of *color science*. [3]

In Euclidean three-space, the distance between points (x_1, y_1, z_1) and (x_2, y_2, z_2) is

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Thus in our problem, the similarity distance between two colors C and S follows as:

$$similarity\_distance = \sqrt{(C_{Red} - S_{Red})^2 + (C_{Green} - S_{Green})^2 + (C_{Blue} - S_{Blue})^2}$$

We then follow by normalizing this metric distance value with the following algorithm

$$degree\_of\_similarity = 1 - \frac{similarity\_distance}{\sqrt{3 * 255^2}}$$

so that the resulting values fall between the *real number* interval of $[0, 1]$.

As one will notice the denominator of the function is the highest possible distance in the metric space. Real world correspondence of this scenario consists the color pair of *Black* and *White*. The algorithm would return 0 as a result for this case. Moreover as expected, the pairing of any color with itself would result the algorithm computing 1 as the similarity degree.

1.1.1.2 Evaluated Similarity

The gist of our methodology for evaluating the similarity degree between two concepts in [1] is conserved in the practical case with minor modifications.

The predicate trees are relatively simple in structural terms. All are depth one, have three children.

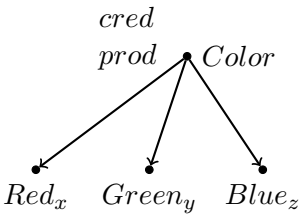


FIGURE 1.1: Example of a color predicate tree

As we know main focus of our algorithm is utilizing predefined similarity relations between the subconcepts. In this particular example, we assume the only predefined similarity relations are the ones between the same RGB main colors. So for every Red_α and Red_β a similarity relations is defined where α and β are two integer values from the interval $[0, 255]$.

The corresponding function is as follows:

$$similarity\_single\_color = 1 - \frac{|c_1 - c_2|}{255}$$

which informally equals to the normalization of the metric distance on a single coordinate of the colors, so that it falls into the interval of $[0, 1]$.

Lastly, the evaluation function follows the same procedure described in [1] .

1.1.1.3 Pseudo Credibility Values

The main focus of interest is finding the weight constants in the *Hybrid Approach(HA)* in an automatic sense. As one might observe, the problem consists the necessary information for calculating the values of *VA* and *EA* thus for fixing the weights, solely the credibility value should also be known. So for training weights, there is a need for some initial *pseudo credibility values*.

In the end we want the product of our similarity and credibility estimation, close to the real degree of similarity.

$$degree\_of\_similarity \times degree\_of\_credibility \cong real\_similarity$$

Through the preceding sections, we know evaluating both of values concerning similarities. So by putting these values in the equation, we may compute the variable of interest.

1.1.1.4 Computing Weights

In [1] , we introduce the formula for the *Hybrid Approach* as follows:

$$\mathit{Credibility} = (w \times [\textbf{Vertex\_Approach}]) + ((1 - w) \times [\textbf{Edge\_Approach}]) \tag{1.1}$$

where $w \in [0..1]$.

As mentioned earlier, when one has the value of credibility, getting the value of the weights from the formula is trivial.

One noteworthy concept here is the *Global Weight*. The focus of this methodology is training the weights from some pre-given info, then fixing the weights through some procedure so the credibility values for new fuzzy rules can be determined.

For maintaining the Multi-Adjoint properties as shown in [2], we adopt the minimum operator for evaluating the global weight. Informally, we train a single weight for every similarity pair in our training set, then set the minimum of those values as our global weight.

1.1.1.5 The Credibility

When the weights are computed, the credibility values are easily evaluated via the *Hybrid Approach* evaluation function. This consists the last step of the methodology, which could just be followed by checking both the credibility value and the similarity value together in comparison to the real degree of similarity.

Here the desired outcome is getting a conservative results, so to speak a value which does not exceed the original degree of similarity, but still finding one which is relatively close.

1.2 A Practical Example

Suppose we have the following six color pairings in our training set:

```
Salmon Coral
Tomato Peru
YellowGreen MediumAquamarine
FineBrick LightSeaGreen
Crimson RoyalBlue
Snow MidnightBlue
```

LISTING 1.1: The training set

For the sake of an interesting display, let us select a pairing from the training set and one completely outside of the initial knowledge base, in order to construct our test set:

```
Tomato Peru
Pink Violet
```

LISTING 1.2: The training set

With these information in hand, we are ready for pursuing with the calculations.

1.2.1 Training Process for Salmon and Coral

In this section we show step by step the process of training a weight via utilizing the info of a color pairing from the training set. The task is simply following the explicit process described in section 1.1.1.

The first step is finding the real-world similarity proximity between the colors, namely *Salmon* and *Coral*. And for that, one needs to find out the metric distance of similarity of the colors.

We input the corresponding *RGB* values of the colors into the modified euclidian function, defined in section 1.1.1.1.

$$similarity\_distance = \sqrt{(C_{Red} - S_{Red})^2 + (C_{Green} - S_{Green})^2 + (C_{Blue} - S_{Blue})^2}$$

$$\begin{aligned} similarity\_distance &= \sqrt{(C_{Red} - S_{Red})^2 + (C_{Green} - S_{Green})^2 + (C_{Blue} - S_{Blue})^2} \\ &= \sqrt{(C_{250} - S_{255})^2 + (C_{128} - S_{127})^2 + (C_{114} - S_{80})^2} \\ &\cong \mathbf{34.38} \end{aligned} \tag{1.2}$$

Then use this metric distance value in the similarity evaluation function:

$$\begin{aligned} degree\_of\_similarity &= 1 - \frac{similarity\_distance}{\sqrt{3 * 255^2}} \\ &= 1 - \frac{34.48}{\sqrt{3 * 255^2}} \\ &\cong \mathbf{92.21} \end{aligned} \tag{1.3}$$

Second step of the process is finding the degree of similarity between the colors via using our own evaluation algorithm. As we know first thing to do here is handling with the calculation of the similarity proximities of subconcepts. There will be three such relations that have to be taken into account. Let us compute one of them, namely the one concerning the *Red* value of the colors:

$$\begin{aligned} similarity\_single\_color &= 1 - \frac{|c_1 - c_2|}{255} \\ &= 1 - \frac{|250 - 255|}{255} \\ &\cong \mathbf{0.98} \end{aligned} \tag{1.4}$$

After doing the same computation for the subconcept similarity relations regarding *Green* and *Blue*, and putting them in the evaluation function, we get:

$$degree\_of\_similarity = 0.947$$

In section 1.1.1.3 we have stated that one we have the both results of the similarity evaluations, we may conclude the credibility value which will be used for training the weights. The corresponding formula and the computation is as follows:

$$\begin{aligned} degree\_of\_similarity \times degree\_of\_credibility &= real\_similarity \\ degree\_of\_credibility &= \frac{real\_similarity}{degree\_of\_similarity} \\ &= 1 - \frac{0.947}{0.98} \\ &\cong \mathbf{0.973} \end{aligned} \tag{1.5}$$

The last computation enables us to continue with the last step of training, which is evaluating the weight. In addition to the credibility value, the *VA* and *EA* values should also be calculated. Since all of the subconcepts are included in some similarity relation,

the *VA* value is 1. Moreover as there are three predefined subconcept similarity relations out of nine possible relations(i.e. $3 \times 3 = 9$), the corresponding *EA* value is $1/3$.

With the help of all these information, the so-called *trained weight value* is:

$$\begin{aligned} \textit{Credibility} &= (w \times [\textbf{Vertex\_Approach}]) + ((1-w) \times [\textbf{Edge\_Approach}]) \\ w &= \frac{\textit{Credibility} - [\textbf{Edge\_Approach}]}{[\textbf{Vertex\_Approach}] - [\textbf{Edge\_Approach}]} \\ &= \frac{0.973 - 0.33}{1 - 0.33} \\ &\cong \mathbf{0.959} \end{aligned} \tag{1.6}$$

And this concludes our calculation for the first color pair of the training set.

1.2.2 The Outcome

A complete implementation of the process in *C++* programming language is displayed at the appendix.

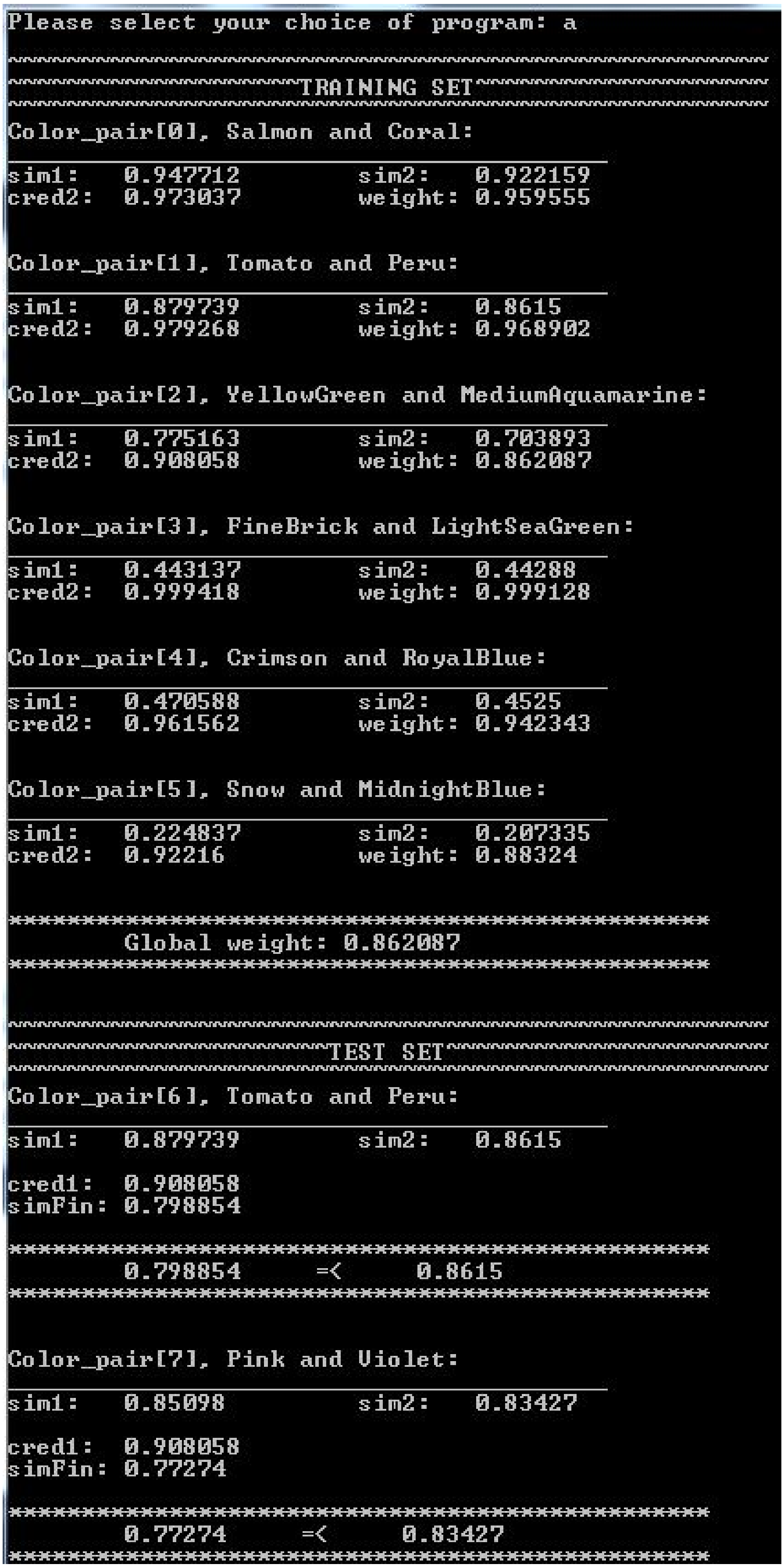
Regarding this example, the console output of the program is depicted in the following page.

As stated in section 1.1.1.4, in order to preserve *Multi-Adjoint* properties, we take the *minumum* operation as the merging tool of the weight values from the training set.

By multiplying the credibility and the similarity values evaluated from our algorithm, we compare the resulting number with the real similarity degree.

As can be seen in both examples, particularly both in the case of the example from the training set and also the newly introduced color pair, the results prove to be conservative. In other words they do not exceed the real values, which is a desired feature. In addition to that, we observe that the evaluated values are relatively close to the real-word correspondences.

As both of these features are satisfied by the result of our methodology where a relatively small knowledge base is utilized, we consider our method to be displaying extreme promise for real world applications.



```
Please select your choice of program: a
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~TRAINING SET~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Color_pair[0], Salmon and Coral:
__________________________________________________
sim1:   0.947712        sim2:   0.922159
cred2:  0.973037        weight: 0.959555


Color_pair[1], Tomato and Peru:
__________________________________________________
sim1:   0.879739        sim2:   0.8615
cred2:  0.979268        weight: 0.968902


Color_pair[2], YellowGreen and MediumAquamarine:
__________________________________________________
sim1:   0.775163        sim2:   0.703893
cred2:  0.908058        weight: 0.862087


Color_pair[3], FineBrick and LightSeaGreen:
__________________________________________________
sim1:   0.443137        sim2:   0.44288
cred2:  0.999418        weight: 0.999128


Color_pair[4], Crimson and RoyalBlue:
__________________________________________________
sim1:   0.470588        sim2:   0.4525
cred2:  0.961562        weight: 0.942343


Color_pair[5], Snow and MidnightBlue:
__________________________________________________
sim1:   0.224837        sim2:   0.207335
cred2:  0.92216         weight: 0.88324


************************************************************
        Global weight: 0.862087
************************************************************


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~TEST SET~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Color_pair[6], Tomato and Peru:
__________________________________________________
sim1:   0.879739        sim2:   0.8615

cred1:  0.908058
simFin: 0.798854

************************************************************
        0.798854     =<     0.8615
************************************************************


Color_pair[7], Pink and Violet:
__________________________________________________
sim1:   0.85098         sim2:   0.83427

cred1:  0.908058
simFin: 0.77274

************************************************************
        0.77274     =<     0.83427
************************************************************
```

Chapter 2

Implementation

Here we display the implementation of the methodology that is discussed in detail in chapter 1.

Firstly we should briefly mention about the flow of the program and the structure of the input files.

The main program contains of two subprograms. The first one of those takes two input files, namely *colorNames.txt* and *colors.txt*. The program takes a used-defined number of color pairs. The last two pairs are the test set, and the rest of the pairs consist the training set. One of the two pairings of the test set is expected to be from the training set, and the other one outside of the domain of the training set. So the program trains the weights via the color pairings from the training set and evaluate the credibility value of the one from the test set. For this regard, *colorNames.txt* store the information concerning the names of the colors. Every line displays one pair, and in each pair colors are separated by the *space* escape character. In a similar manner, in each line *colors.txt* saves the *RGB* values of the corresponding colors from the other input file. The six values, i.e. three for each of the colors, are separated by *space* characters and lines are split with an *enter key*.

An example case for the input files is displayed as follows:

```
250 128 114 255 127 80
255 99 71 210 105 30
154 205 50 102 205 170
178 34 34 32 178 170
220  20  60 65 105 225
255 250 250 25 25 112
255 99 71 210 105 30
255 192 203 238 130 238
```

Listing 2.1: colors.txt

```
Salmon Coral
Tomato Peru
YellowGreen MediumAquamarine
FineBrick LightSeaGreen
Crimson RoyalBlue
Snow MidnightBlue
Tomato Peru
Pink Violet
```

Listing 2.2: colorNames.txt

The second subprogram works with a single input file, namely *colorsAll.txt*. The input file contains all the information regarding the complete set of colors in *X11* scheme. At each line you have the name and the *RGB* values of a particular color.

The subprogram asks the user two name two colors from the domain. It then takes out those two values and their associated information from the knowledge base. All of the possible pairings between the rest of the colors are computed by the program, which exactly corresponds to 9591 pairings. With this huge amount of information, the weights are trained and later utilized for evaluating the credibility degree of the similarity value concerning the color pairing of interest.

You may observe an example instance of the *colorsAll.txt* input file. That is followed by two example outputs of the program for the same color pair, but with different methodologies of evaluating the global weight, namely *minimum rule* and *average rule*. And finally you may inspect the complete implementation of the methodology in *C++* programming language.

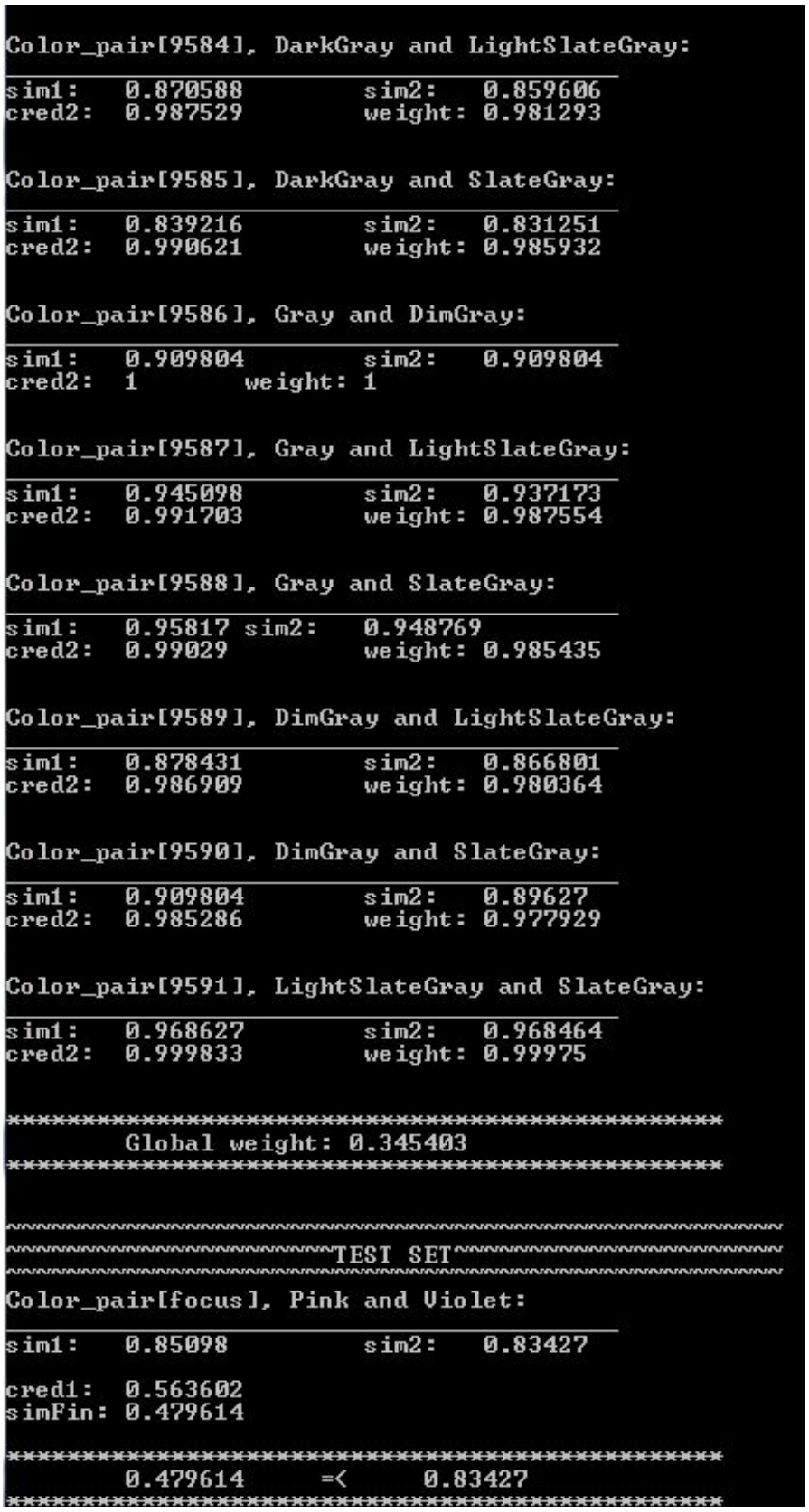
```
IndianRed 205 92 92
LightCoral 240 128 128
Salmon 250 128 114
DarkSalmon 233 150 122
LightSalmon 255 160 122
Red 255 0 0
Crimson 220 20 60
FireBrick 178 34 34
DarkRed 139 0 0
Pink 255 192 203
LightPink 255 182 193
HotPink 255 105 180
DeepPink 255  20 147
MediumVioletRed 199  21 133
PaleVioletRed 219 112 147
LightSalmon 255 160 122
Coral 255 127  80
```

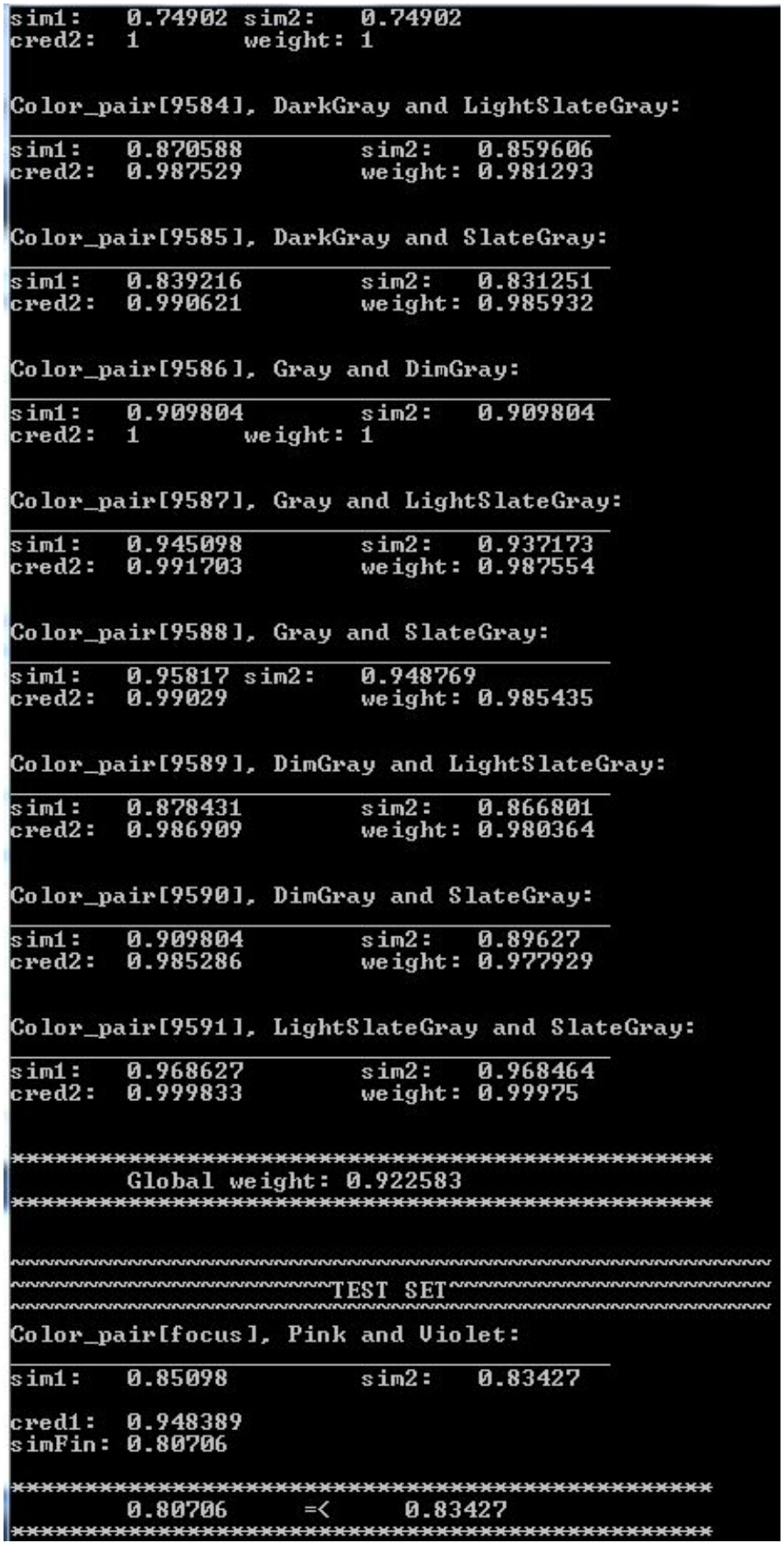
```
Tomato 255  99  71
OrangeRed 255  69   0
DarkOrange 255 140   0
Orange 255 165   0
Gold 255 215   0
Yellow 255 255   0
LightYellow 255 255 224
LemonChiffon 255 250 205
LightGoldenrodYellow 250 250 210
PapayaWhip 255 239 213
Moccasin 255 228 181
PeachPuff 255 218 185
PaleGoldenrod 238 232 170
Khaki 240 230 140
DarkKhaki 189 183 107
Lavender 230 230 250
Thistle 216 191 216
Plum 221 160 221
Violet 238 130 238
Orchid 218 112 214
Fuchsia 255   0 255
Magenta 255   0 255
MediumOrchid 186  85 211
MediumPurple 147 112 219
BlueViolet 138  43 226
DarkViolet 148   0 211
DarkOrchid 153  50 204
DarkMagenta 139   0 139
Purple 128   0 128
Indigo  75   0 130
DarkSlateBlue 72  61 139
SlateBlue 106  90 205
MediumSlateBlue 123 104 238
GreenYellow 173 255  47
Chartreuse 127 255   0
LawnGreen 124 252   0
Lime 0 255   0
LimeGreen  50 205  50
PaleGreen 152 251 152
LightGreen 144 238 144
MediumSpringGreen 0 250 154
SpringGreen 0 255 127
MediumSeaGreen 60 179 113
SeaGreen 46 139  87
ForestGreen 34 139  34
Green 0 128   0
DarkGreen 0 100   0
YellowGreen 154 205  50
OliveDrab 107 142  35
Olive 128 128   0
DarkOliveGreen 85 107  47
```

```
MediumAquamarine 102 205 170
DarkSeaGreen 143 188 143
LightSeaGreen 32 178 170
DarkCyan  0 139 139
Teal  0 128 128
Aqua 0 255 255
Cyan 0 255 255
LightCyan 224 255 255
PaleTurquoise 175 238 238
Aquamarine 127 255 212
Turquoise 64 224 208
MediumTurquoise 72 209 204
DarkTurquoise 0 206 209
CadetBlue 95 158 160
SteelBlue 70 130 180
LightSteelBlue 176 196 222
PowderBlue 176 224 230
LightBlue 173 216 230
SkyBlue 135 206 235
LightSkyBlue 135 206 250
DeepSkyBlue 0 191 255
DodgerBlue 30 144 255
CornflowerBlue 100 149 237
RoyalBlue 65 105 225
Blue 0   0 255
MediumBlue 0   0 205
DarkBlue 0   0 139
Navy 0   0 128
MidnightBlue 25  25 112
Cornsilk 255 248 220
BlanchedAlmond 255 235 205
Bisque 255 228 196
NavajoWhite 255 222 173
Wheat 245 222 179
BurlyWood 222 184 135
Tan 210 180 140
RosyBrown 188 143 143
SandyBrown 244 164  96
Goldenrod 218 165  32
DarkGoldenrod 184 134  11
Peru 205 133  63
Chocolate 210 105  30
SaddleBrown 139  69  19
Sienna 160  82  45
Brown 165  42  42
Maroon 128   0   0
White 255 255 255
Snow 255 250 250
Honeydew 240 255 240
MintCream 245 255 250
Azure 240 255 255
```

```
AliceBlue 240 248 255
GhostWhite 248 248 255
WhiteSmoke 245 245 245
Seashell 255 245 238
Beige 245 245 220
OldLace 253 245 230
FloralWhite 255 250 240
Ivory 255 255 240
AntiqueWhite 250 235 215
Linen 250 240 230
LavenderBlush 255 240 245
MistyRose 255 228 225
Gainsboro 220 220 220
LightGrey 211 211 211
Silver 192 192 192
DarkGray 169 169 169
Gray 128 128 128
DimGray 105 105 105
LightSlateGray 119 136 153
SlateGray 112 128 144
DarkSlateGray  47  79  79
Black 0   0   0
```

Listing 2.3: colorsAll.txt


Color_pair[9584], DarkGray and LightSlateGray:
sim1: 0.870588 sim2: 0.859606
cred2: 0.987529 weight: 0.981293
Color_pair[9585], DarkGray and SlateGray:
sim1: 0.839216 sim2: 0.831251
cred2: 0.990621 weight: 0.985932
Color_pair[9586], Gray and DimGray:
sim1: 0.909804 sim2: 0.909804
cred2: 1 weight: 1
Color_pair[9587], Gray and LightSlateGray:
sim1: 0.945098 sim2: 0.937173
cred2: 0.991703 weight: 0.987554
Color_pair[9588], Gray and SlateGray:
sim1: 0.95817 sim2: 0.948769
cred2: 0.99029 weight: 0.985435
Color_pair[9589], DimGray and LightSlateGray:
sim1: 0.878431 sim2: 0.866801
cred2: 0.986909 weight: 0.980364
Color_pair[9590], DimGray and SlateGray:
sim1: 0.909804 sim2: 0.89627
cred2: 0.985286 weight: 0.977929
Color_pair[9591], LightSlateGray and SlateGray:
sim1: 0.968627 sim2: 0.968464
cred2: 0.999833 weight: 0.99975
Global weight: 0.345403
TEST SET
Color_pair[focus], Pink and Violet:
sim1: 0.85098 sim2: 0.83427
cred1: 0.563602
simFin: 0.479614
0.479614 =< 0.83427


sim1: 0.74902 sim2: 0.74902
cred2: 1 weight: 1
Color_pair[9584], DarkGray and LightSlateGray:
sim1: 0.870588 sim2: 0.859606
cred2: 0.987529 weight: 0.981293
Color_pair[9585], DarkGray and SlateGray:
sim1: 0.839216 sim2: 0.831251
cred2: 0.990621 weight: 0.985932
Color_pair[9586], Gray and DimGray:
sim1: 0.909804 sim2: 0.909804
cred2: 1 weight: 1
Color_pair[9587], Gray and LightSlateGray:
sim1: 0.945098 sim2: 0.937173
cred2: 0.991703 weight: 0.987554
Color_pair[9588], Gray and SlateGray:
sim1: 0.95817 sim2: 0.948769
cred2: 0.99029 weight: 0.985435
Color_pair[9589], DimGray and LightSlateGray:
sim1: 0.878431 sim2: 0.866801
cred2: 0.986909 weight: 0.980364
Color_pair[9590], DimGray and SlateGray:
sim1: 0.909804 sim2: 0.89627
cred2: 0.985286 weight: 0.977929
Color_pair[9591], LightSlateGray and SlateGray:
sim1: 0.968627 sim2: 0.968464
cred2: 0.999833 weight: 0.99975
Global weight: 0.922583
TEST SET
Color_pair[focus], Pink and Violet:
sim1: 0.85098 sim2: 0.83427
cred1: 0.948389
simFin: 0.80706
0.80706 =< 0.83427

```
#include<iostream>
using namespace std;
#include<string>
#include<math.h>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>

struct color{
        int x;                  //first coordinate
        int y;                  //second coordinate
        int z;                  //third coordinate
        string c;               //the color name
};

float euqDist(float, float, float, float, float, float);                //
    euclidian distance between two colors.
float simOneColor(float, float);
                                    //similarity between two instances of
    the same color.
float simByED(float);
                                            //similarity degree of two
    colors via euqlidian distance.
float simByAlg(float, float, float, float, float, float);       //
    similarity degree of two colors via our algorithm.
float credBySim(float, float);
                            //credibility value defined from proximity of
    similarity values.
float weightEval(float, float= 1, float = (1/3.0));
           //weight w, evaluated from cred function.
float credEval(float, float= 1, float = (1/3.0));
           //final credibility value, computed via global weight
void inputHandle(ifstream& f,  vector<vector<float>> &c, string fileName);
void inputColor(ifstream& f,  vector<string> &col, string fileName);
void inputComplete(ifstream& f,  vector<color> &c, string fileName);

int main()
{
        string pc;
        begin:
        cout<<"Please select your choice of program: ";
        cin>>pc;

        if(pc == "1" || pc == "A" || pc == "a"){
                ifstream f1, f2;
                vector<vector<float>> colorPairs;
                inputHandle(f1, colorPairs, "colors.txt");
                vector<string> colorNames;
```

```
            inputColor(f2, colorNames, "colorNames.txt");

            cout<<"\n
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"~~~~~~~~~~~~~~~~~~~~TRAINING SET
~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

            float weightMin = 1;
            for(int i = 0; i < colorPairs.size() - 2; i++){
                        float sim1 = simByAlg(colorPairs[i][0],
colorPairs[i][1], colorPairs[i][2], colorPairs[i][3], colorPairs[i][4],
 colorPairs[i][5]);
                        float sim2 = simByED(euqDist(colorPairs[i
][0], colorPairs[i][1], colorPairs[i][2], colorPairs[i][3], colorPairs[
i][4], colorPairs[i][5]));
                        float cred2 = credBySim(sim1, sim2);
                        float weight = weightEval(cred2);
                        if(weight < weightMin)
                                weightMin = weight;

                        cout<<"Color_pair["<<i<<"], "<<colorNames[i
*2]<<" and "<<colorNames[i*2+1]<<": "<<endl;
                        cout<<"
----------------------------------------"<<endl;
                        cout<<"sim1:\t"<<sim1<<"\tsim2:\t"<<sim2<<"
\ncred2:\t"<<cred2<<" \tweight:\t"<<weight<<"\n\n\n";
            }

            cout<<"**********************************************\n";
            cout<<"\tGlobal weight: "<<weightMin<<endl;
            cout<<"**********************************************\n\n
\n";

            cout<<"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"~~~~~~~~~~~~~~~~~~~~~~TEST SET
~~~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

            int s = colorPairs.size();
            for(int t = s -2; t < s; t++){
                        float sim1 = simByAlg(colorPairs[t][0], colorPairs[
t][1], colorPairs[t][2], colorPairs[t][3], colorPairs[t][4], colorPairs
[t][5]);
                        float sim2 = simByED(euqDist(colorPairs[t][0],
colorPairs[t][1], colorPairs[t][2], colorPairs[t][3], colorPairs[t][4],
 colorPairs[t][5]));
```

```
                    cout<<"Color_pair["<<t<<"], "<<colorNames[t*2]<<"
and "<<colorNames[t*2+1]<<": "<<endl;
                    cout<<"----------------------------------------"<<
endl;
                    cout<<"sim1:\t"<<sim1<<" \tsim2:\t"<<sim2<<"\n\n";

                    float cred1 = credEval(weightMin);      //global
weight is used for finding the final credibility value
                    cout<<"cred1:\t"<<cred1<<endl;
                    float simFin = cred1 * sim1;
                    cout<<"simFin:\t"<<simFin;

                    cout<<"\n\n
**********************************************\n";
                    cout<<"\t"<<simFin<<"      =<      "<<sim2<<endl;
                    cout<<"
**********************************************\n\n\n";
            }
    }

    else if(pc == "2" || pc == "B" || pc == "b"){
            ifstream f3;
            vector<color> col;
            inputComplete(f3, col, "colorsAll.txt");
            string userColor1, userColor2;

            cout<<"\nPlease choose two colors: ";
            cin>>userColor1;
            cin>>userColor2;

            color temp1, temp2;
            for(int i = 0; i < col.size(); i++){
                    if(col[i].c == userColor1){
                            temp1.c = col[i].c;
        //store the color at a temp variable
                            temp1.x = col[i].x;
                            temp1.y = col[i].y;
                            temp1.z = col[i].z;

                            col[i].c = col[col.size()-1].c; //switch it
 with the last element of
                            col[i].x = col[col.size()-1].x; //the
vector in order to pop it out
                            col[i].y = col[col.size()-1].y; //of the
list
                            col[i].z = col[col.size()-1].z;
                            col.pop_back();
                    }
            }

            for(int i = 0; i < col.size(); i++){
```

```
                    if(col[i].c == userColor2){
                            temp2.c = col[i].c;
        //store the color at a temp variable
                            temp2.x = col[i].x;
                            temp2.y = col[i].y;
                            temp2.z = col[i].z;

                            col[i].c = col[col.size()-1].c; //switch it
 with the last element of
                            col[i].x = col[col.size()-1].x; //the
vector in order to pop it out
                            col[i].y = col[col.size()-1].y; //of the
list
                            col[i].z = col[col.size()-1].z;
                            col.pop_back();
                    }
            }

            cout<<"\n Weights evaluated via Min or Average rule: ";
            cin>>pc;

            float weightGlo = 1.0;          //global weight
            if(pc == "1" || pc == "m" || pc == "M"){
                    color t1, t2;
                    int pn =1;                                          //
pair counter

                    for(int i = 0; i < col.size(); i++){
                            for(int j = i+1; j < col.size(); j++){
                                    float sim1 = simByAlg(col[i].x, col
[i].y, col[i].z, col[j].x, col[j].y, col[j].z);
                                    float sim2 = simByED(euqDist(col[i
].x, col[i].y, col[i].z, col[j].x, col[j].y, col[j].z));
                                    float cred2 = credBySim(sim1, sim2)
;
                                    float weight = weightEval(cred2);
                                    if(weight < weightGlo)
                                            weightGlo = weight;

                                    cout<<"Color_pair["<<pn<<"], "<<col
[i].c<<" and "<<col[j].c<<": "<<endl;
                                    cout<<"
----------------------------------------"<<endl;
                                    cout<<"sim1:\t"<<sim1<<"\tsim2:\t"
<<sim2<<"\ncred2:\t"<<cred2<<" \tweight:\t"<<weight<<"\n\n\n";
                                    pn++;
                            }
                    }
            }

            else if(pc == "a" || pc == "A" || pc == "2"){
```

```
                    color t1, t2;
                    vector<float> weights;
                    int pn =1;                                          //
pair counter
                    for(int i = 0; i < col.size(); i++){
                            for(int j = i+1; j < col.size(); j++){
                                    float sim1 = simByAlg(col[i].x, col
[i].y, col[i].z, col[j].x, col[j].y, col[j].z);
                                    float sim2 = simByED(euqDist(col[i
].x, col[i].y, col[i].z, col[j].x, col[j].y, col[j].z));
                                    float cred2 = credBySim(sim1, sim2)
;
                                    float weight = weightEval(cred2);
                                    weights.push_back(weight);

                                    cout<<"Color_pair["<<pn<<"], "<<col
[i].c<<" and "<<col[j].c<<": "<<endl;
                                    cout<<"
---------------------------------------"<<endl;
                                    cout<<"sim1:\t"<<sim1<<"\tsim2:\t"
<<sim2<<"\ncred2:\t"<<cred2<<" \tweight:\t"<<weight<<"\n\n\n";
                                    pn++;
                            }
                    }
                    for(int i = 0; i < col.size(); i++)
                            weightGlo += weights[i];
                    weightGlo /= col.size();
            }

            cout<<"*********************************************\n";
            cout<<"\tGlobal weight: "<<weightGlo<<endl;
            cout<<"*********************************************\n\n
\n";

            cout<<"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"~~~~~~~~~~~~~~~~~~~~~~TEST SET
~~~~~~~~~~~~~~~~~~~~~~\n";
            cout<<"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

            float sim1 = simByAlg(temp1.x, temp1.y, temp1.z, temp2.x,
temp2.y, temp2.z);
            float sim2 = simByED(euqDist(temp1.x, temp1.y, temp1.z,
temp2.x, temp2.y, temp2.z));
            cout<<"Color_pair[focus], "<<temp1.c<<" and "<<temp2.c<<":
"<<endl;
            cout<<" ---------------------------------------"<<endl;
            cout<<"sim1:\t"<<sim1<<" \tsim2:\t"<<sim2<<"\n\n";
```

```
                float cred1 = credEval(weightGlo);      //global weight is
    used for finding the final credibility value
                cout<<"cred1:\t"<<cred1<<endl;
                float simFin = cred1 * sim1;
                cout<<"simFin:\t"<<simFin;

                cout<<"\n\n
    ********************************************\n";
                cout<<"\t"<<simFin<<"      =<      "<<sim2<<endl;
                cout<<"********************************************\n\n
    \n";
        }

        else{
                cout<<"Wrong command!\n";
                goto begin;
        }

        getchar();
        getchar();
        return 0;
}

//euclidian distance between two colors
float euqDist(float a, float b, float c, float x, float y, float z)
{
        return sqrt(pow(fabs(a-x),2)+pow(fabs(b-y),2)+pow(fabs(c-z),2));
}

//similarity between two instances of the same color. (ex: red_90 and
    red_187)
float simOneColor(float c1, float c2)
{
        return 1-((fabs(c1-c2))/255);
}

//similarity degree of two colors via euqlidian distance
float simByED(float dist)
{
        return 1-(dist/(255*sqrt(3.0)));
        /*
        float res = 1-(dist/(255*sqrt(3.0)));
        if(res < 0.1)
                return 0;
        else
                return res;*/
}

//similarity degree of two colors via our algorithm
float simByAlg(float a, float b, float c, float x, float y, float z)
{
```

```
        return (((simOneColor(a, x))+(simOneColor(b, y))+(simOneColor(c, z)
    ))/3);
}

//credibility value defined from proximity of similarity values
float credBySim(float sim1, float sim2)
{
        //return 1-(fabs(sim1 - sim2));
        float sim = (sim2+0.01) / (sim1+0.01);  //in order to prevent
    division by zero
        if(sim > 1)
                return 1;
        else
                return sim;
}

//weight w, evaluated from cred function
float weightEval(float cred, float va, float ea)
           //default values for va and ea are 1 and 1/3 respectively
{
                                                                    //since
     those are the most common cases, but could be overwritten
        return ((cred-ea)/(va-ea));
}

float credEval(float weight, float va, float ea)
           //again default values for va and ea are 1 and 1/3 respectively
{
        return ((weight * va) + ((1-weight) * ea));
}


void inputHandle(ifstream& f,  vector<vector<float>> &c, string fileName)
{
        string temp;
        f.open(fileName);

        if (!f) {
        cout << "Unable to open file";
        cin>>temp;
        exit(1); // terminate with error
        }

        int i = 0;
        char line[1024];


        while(f.getline(line, 1024)){
                istringstream ss(line, istringstream::in);
                float rate;
                vector<float> temp;
```

```
                c.push_back(temp);

                for(int j = 0; j < 6; j++){
                        ss >> rate;
                        c[i].push_back(rate);
                }
                i++;
        }

        f.close();
}

void inputColor(ifstream& f,  vector<string> &c, string fileName)
{
        string temp;
        f.open(fileName);

        if (!f) {
        cout << "Unable to open file";
        cin>>temp;
        exit(1); // terminate with error
        }

        char line[1024];
        string tempColor;

        while(f.getline(line, 1024)){
                istringstream ss(line, istringstream::in);
                ss >> tempColor;                                    //first
    color of the pairing
                c.push_back(tempColor);
                ss >> tempColor;                                    //second
    color of the pairing
                c.push_back(tempColor);
        }

        f.close();
}

void inputComplete(ifstream& f,  vector<color> &c, string fileName)
{
        string temp;
        f.open(fileName);

        if (!f) {
        cout << "Unable to open file";
        cin>>temp;
        exit(1); // terminate with error
        }
```

```
        char line[1024];

        while(f.getline(line, 1024)){
                istringstream ss(line, istringstream::in);
                color temp;

                ss >> temp.c;
                ss >> temp.x;
                ss >> temp.y;
                ss >> temp.z;

                c.push_back(temp);

                cout<<temp.x<<" \t"<<temp.y<<" \t"<<temp.z<<" \t"<<temp.c<<
    endl;
        }

        f.close();
}
```

LISTING 2.4: The C++ implementation of the application concerning colors

Bibliography

[1] Sinan Egilmez. A sound and efficient approach for the similarity concept in fuzzy logic. Master's thesis, Facultad de Informatica, 2012.

[2] Susana Munoz-Hernandez, Victor Pablos-Ceruelo, and Hannas Strass. Rfuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over prolog. 2010.

[3] Gaurav Sharma. *Digital Color Imaging Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 2002.

[4] Craig S. Wright. *The IT Regulatory and Standards Compliance Handbook:: How to Survive Information Systems Audit and Assessments*. Syngress Publishing, 2008.