

# Proyecto de Lógica Difusa Simulación

Dalianys Pérez Perera  
C-411

# Índice

|   |           |
|---|-----------|
| <b>1. Introducción</b>                                      | <b>3</b>  |
| <b>2. Características del Sistema de Inferencia</b>         | <b>3</b>  |
| <b>3. Principales ideas seguidas para la implementación</b> | <b>3</b>  |
| 3.1. Variables . . . . .                                    | 4         |
| 3.2. Conjuntos difusos . . . . .                            | 5         |
| 3.3. Operaciones entre conjuntos . . . . .                  | 5         |
| 3.4. Reglas . . . . .                                       | 6         |
| 3.5. Funciones de agregación . . . . .                      | 6         |
| <b>4. Propuesta de problema</b>                             | <b>6</b>  |
| 4.1. Consultas realizadas . . . . .                         | 7         |
| <b>5. Consideraciones obtenidas</b>                         | <b>12</b> |

# 1. Introducción

Los problemas que pretenden reflejar un comportamiento medianamente realista suelen requerir un elevado número de reglas mientras que los problemas simples no siempre llegan a tener aplicaciones en el mundo cotidiano, etc. Con este proyecto se ofrece un ejemplo que muestre el comportamiento de un sistema con reglas básicas y sencillas para una aplicación real.

Se ha definido un sistema difuso que controla un semáforo, del que se describirán sus entradas, salidas, reglas, operadores de inferencia así como detalles de implementación.

# 2. Características del Sistema de Inferencia

El sistema propuesto consta de funciones de pertenencia *triangulares* y *trapezoidales*. También posee los métodos de agregación de **Mamdani** y **Larsen** al igual que todos los métodos de desdifusificación (*Centroide*, *Bisección* y todas las variantes de los *Máximos*)

# 3. Principales ideas seguidas para la implementación

La solución está implementada en el lenguaje de programación *Python* y la misma se divide en varios módulos o componentes:

- `fuzzy_system`: contiene la clase `FuzzySystem` constituyendo la maquinaria de un sistema difuso. La misma ofrece funcionalidades como la adición de una variable al sistema, adición de una regla, la fase de fusificación y de inferencia. Para crear un sistema difuso es necesario definir el método de agregación a utilizar, por tanto, este es recibido en la inicialización de la clase.
- `agregation`: brinda la implementación de los dos métodos de agregación contemplados: *Mamdani* y *Larsen*.
- `defuzzy`: incluye los métodos de desdifusificación implementados: centro de gravedad(`centroid`), bisección(`bisector`) y los métodos de máximos brindados son: menor de los máximos(`left_max`), mayor de los máximos (`right_max`) y media de los máximos(`mean_max`).
- `knowledge`: define lo que serían las variables difusas, las reglas y una función para *parsear* las reglas recibidas como entrada en formato *string*.
- `membership`: contiene las definiciones de las funciones de membresía utilizadas, además de las operaciones que se pueden establecer entre estas como la *unión*, *intersección*, *complemento* y el *corte* de una función dado un *alpha*.

La implementación de las funciones de membresía y las operaciones admisibles entre estas se realizó completamente funcional, a continuación se muestra un segmento de código de dicha implementación.

```

def triangular(a, b, c):
    def f(x):
        if x < a:
            return 0
        elif a <= x and x <= b:
            return 1. * (x - a) / (b - a)
        elif b <= x and x <= c:
            return 1. * (c - x) / (c - b)
        else:
            return 0
    return f

def trapezoidal(a, b, c, d):
    def f(x):
        if x <= a or x >= d:
            return 0
        if x >= b and x <= c:
            return 1
        if x >= a and x <= b:
            return 1. * (x - a) / (b - a)
        return 1. * (d - x) / (d - c)
    return f

def union(f, g):
    def wrapper(x):
        return max(f(x), g(x))
    return wrapper

def intersection(f, g):
    def wrapper(x):
        return min(f(x), g(x))
    return wrapper

def complement(f):
    def wrapper(x):
        return 1 - f(x)
    return wrapper

```

### 3.1. Variables

La declaración de variables se realiza de la siguiente forma:

```

v1 = Variable("Traffic")
v1.add_value("Null", triangular(0, 5, 10), (0, 10) )
v1.add_value("Moderate", trapezoidal(5, 10, 20, 25), (5, 25) )

```

```
v1.add_value("Intense", triangular(15, 25, 35), (15, 35) )
```

Con la función `add_value` se añaden los números difusos a la variable en cuestión, donde el primer parámetro es el tipo de función de membresía y el segundo un par que representa su dominio de definición distinta de cero.

### 3.2. Conjuntos difusos

Como se dijo anteriormente, solo se implementaron conjuntos difusos con función de pertenencia triangulares y trapezoidales.

La función `triangular(x1, x2, x3)` retorna un conjunto difuso donde para elementos menores que  $x1$  o mayores que  $x3$  no pertenecen en absoluto al conjunto, de  $x1$  a  $x2$  aumenta de forma lineal desde pertenencia 0 a 1 y de  $x2$  a  $x3$  disminuye de 1 a 0.

```
f1 = triangular(10, 30, 50)
f2 = triangular(30, 50, 70)
```

La función `trapezoidal(x1, x2, x3, x4)` retorna un conjunto difuso donde para valores menores que  $x1$  o mayores que  $x4$  la pertenencia se define como 0, de  $x1$  a  $x2$  aumenta de 0 a 1 de forma lineal, de  $x2$  a  $x3$  es 1 y de  $x3$  a  $x4$  nuevamente disminuye de 1 a 0.

```
g1 = trapezoidal(10, 20, 30, 40)
g2 = trapezoidal(30, 50, 70, 90)
```

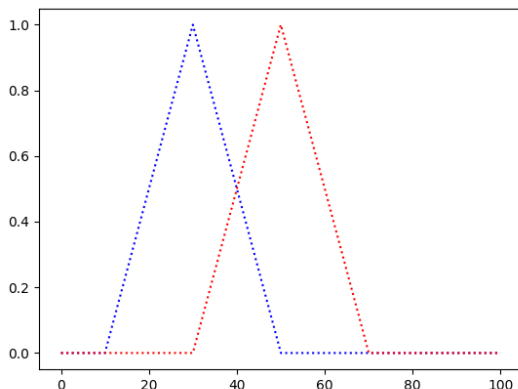


Figura 1: Ejemplo de funciones triangulares

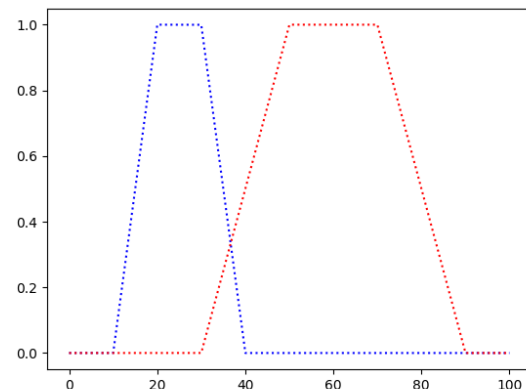


Figura 2: Ejemplo de funciones trapezoidales

### 3.3. Operaciones entre conjuntos

Es posible realizar operaciones entre las funciones definidas, a continuación se muestran algunos ejemplos utilizando las funciones de la Figura 1 y 2.

```
f1_&f2 = intersection(f1, f2)
g1_|g2 = union(g1, g2)
not_f1 = complement(f1)
```

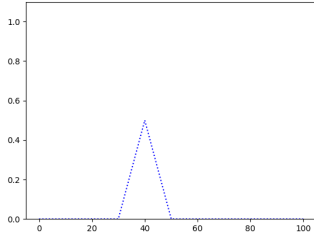


Figura 3: Intersección  $f_1$  y  $f_2$

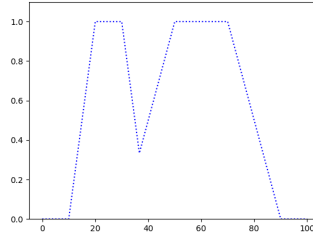


Figura 4: Unión entre  $g_1$  y  $g_2$

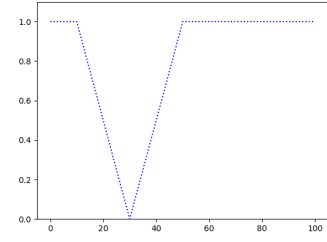


Figura 5: Complemento de  $f_1$

### 3.4. Reglas

Las reglas son recibidas en formato texto y pertenecen a la siguiente gramática:

$$S \rightarrow A' \implies Q$$

$$Q \rightarrow C$$

$$A \rightarrow C \mid C' \text{ and } A$$

$$C \rightarrow Id \text{ 'is' } Id \mid \text{not' } Id \text{ 'is' } Id$$

Por tanto primeramente son *parseadas* a través del método `parse_rule` el cual devuelve una instancia de tipo `Rule`.

### 3.5. Funciones de agregación

Contempladas en el archivo `agregation.py`

```
class Mamdani:
    def __init__(self, defuzzifier, op):

    def predict(self, variables, umbral_corts, rules):

class Larsen:
    def __init__(self, defuzzifier, op):

    def predict(self, variables, umbral_corts, rules):
```

Ambas reciben como parámetro la función que se utilizará para la desfusificación y el operador (*max*, *min*, *product*) que se aplicará durante la evaluación de cada regla a partir de los valores de entrada. El resultado de esta evaluación constituye el *alpha* de corte que aporta cada regla para la evaluación de su consecuente.

## 4. Propuesta de problema

El modelo se basa en el control de un semáforo, específicamente la duración de la luz verde, según la cantidad de peatones que necesitan cruzar la calle y la cantidad de carros detenidos en cola durante la luz roja. En primer lugar se van a definir las variables de entrada

y salida del sistema. El semáforo tiene dos entradas y una salida.

## Entradas

- cantidad de peatones(Walkers): (Low, Medium, High) variable discreta con dominio entre 0 y 20.
- cantidad de carros (Traffic): (Null, Moderate, Intense) variable discreta con dominio entre 0 y 35.

## Salida

- duración luz verde(GreenDuration): (Short, Medium, Long) variable con dominio en los reales entre 0 y 100.

A continuación se visualizan los conjuntos difusos definidos por cada variable y a su vez el tipo de función de membresía de los mismos.

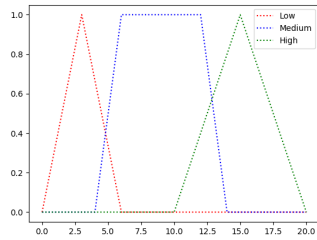


Figura 6: *Walkers*

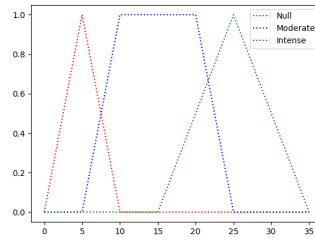


Figura 7: *Traffic*

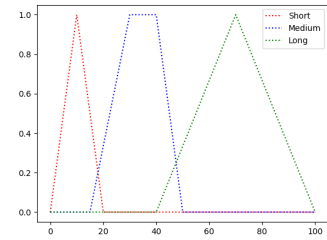


Figura 8: *GreenDuration*

Las reglas de inferencia para este sistema difuso están simplificadas en el cuadro 1 y sus producciones son del tipo:

$$Walkers = A \text{ and } Traffic = B \Rightarrow GreenDuration = C$$

donde A, B y C son conjuntos difusos definidos anteriormente para cada variable.

| Walkers/Traffic | Null   | Moderate | Intense |
|-----------------|--------|----------|---------|
| Low             | Medium | Short    | Long    |
| Medium          | Short  | Medium   | Medium  |
| High            | Short  | Medium   | Long    |

Cuadro 1: Reglas del sistema difuso

## 4.1. Consultas realizadas

A continuación se muestran algunas consultas realizadas al sistema de inferencia propuesto:

**Entrada:**  $Walkers = 7$ ,  $Traffic = 22$   
**Función de agregación:** Mamdani  
**Función de desfusificación:** centroid  
**Operador:** min

**Salida:**  $GreenDuration = 33.2778$

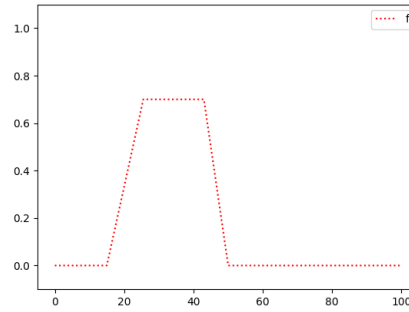


Figura 9: Conjunto difuso resultado de la agregación

**Entrada:**  $Walkers = 7$ ,  $Traffic = 22$   
**Función de agregación:** Mamdani  
**Función de desfusificación:** left\_max  
**Operador:** min

**Salida:**  $GreenDuration = 25.5026$

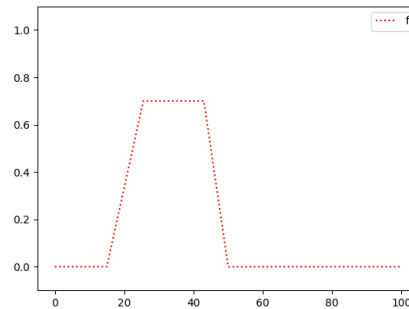


Figura 10: Conjunto difuso resultado de la agregación

**Entrada:**  $Walkers = 7$ ,  $Traffic = 22$   
**Función de agregación:** Larsen  
**Función de desfusificación:** centroid  
**Operador:** min

**Salida:**  $GreenDuration = 33.5185$



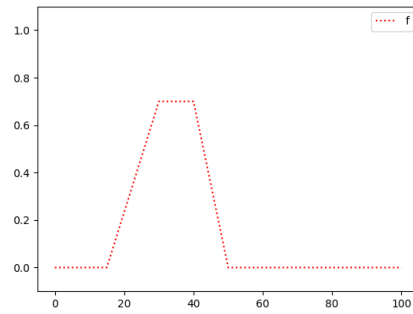


Figura 11: Conjunto difuso resultado de la agregación

**Entrada:**  $Walkers = 7$ ,  $Traffic = 22$

**Función de agregación:** Larsen

**Función de desfusificación:** mean\_max

**Operador:** min

**Salida:**  $GreenDuration = 34.9985$

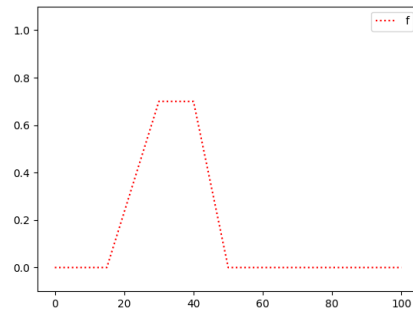


Figura 12: Conjunto difuso resultado de la agregación

**Entrada:**  $Walkers = 8$ ,  $Traffic = 16$

**Función de agregación:** Mamdani

**Función de desfusificación:** bisector

**Operador:** min

**Salida:**  $GreenDuration = 33.75$

**Entrada:**  $Walkers = 4$ ,  $Traffic = 30$

**Función de agregación:** Mamdani

**Función de desfusificación:** centroid

**Operador:** min

**Salida:**  $GreenDuration = 70.0$

**Entrada:**  $Walkers = 4$ ,  $Traffic = 30$

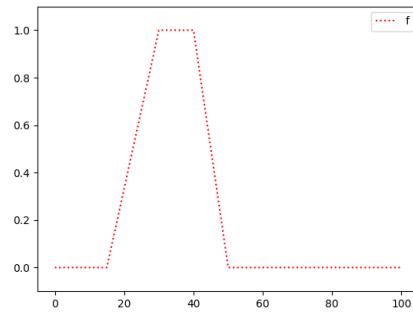


Figura 13: Conjunto difuso resultado de la agregación

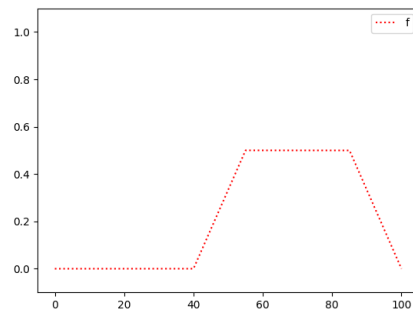


Figura 14: Conjunto difuso resultado de la agregación

**Función de agregación:** Larsen

**Función de desdifusificación:** centroid

**Operador:** min

**Salida:**  $GreenDuration = 70.0$

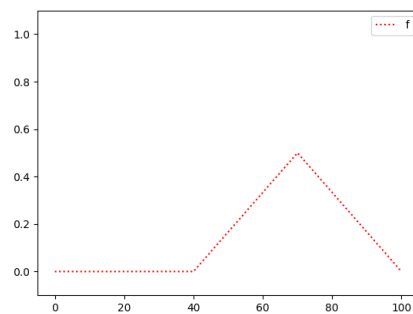


Figura 15: Conjunto difuso resultado de la agregación

**Entrada:**  $Walkers = 18$ ,  $Traffic = 23$

**Función de agregación:** Mandani

**Función de desfusificación:** centroid  
**Operador:** min

**Salida:**  $GreenDuration = 56.1728$

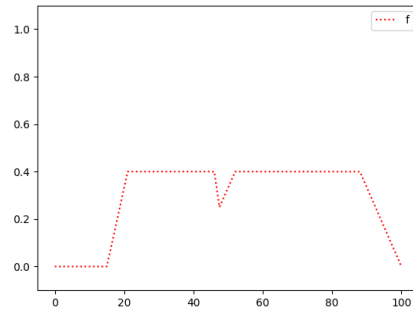


Figura 16: Conjunto difuso resultado de la agregación

**Entrada:**  $Walkers = 3$ ,  $Traffic = 7$   
**Función de agregación:** Mandani  
**Función de desfusificación:** mean\_max  
**Operador:** min

**Salida:**  $GreenDuration = 33.9984$

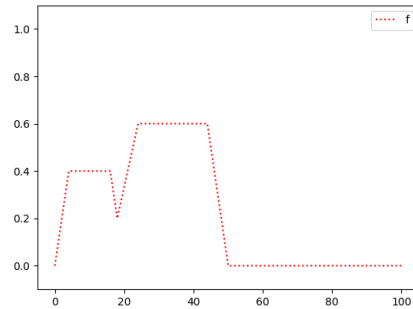


Figura 17: Conjunto difuso resultado de la agregación

**Entrada:**  $Walkers = 8$ ,  $Traffic = 4$   
**Función de agregación:** Larsen  
**Función de desfusificación:** bisector  
**Operador:** min

**Salida:**  $GreenDuration = 10.0$

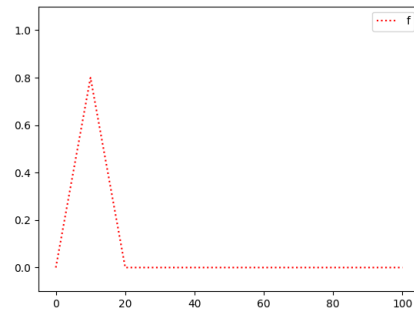


Figura 18: Conjunto difuso resultado de la agregación

## 5. Consideraciones obtenidas

El comportamiento del sistema difuso dependerá de los operadores utilizados durante la fase de fusificación, de la función de composición para computar el consecuente y del método de conversión de difuso a nítido.