BSc. Artificial Intelligence & Data Science

CM 2602
ARTIFICIAL INTELLIGENCE
COURSEWORK REPORT

Module Leader: Mr. Nipuna Senanayake

SANUKA JAYASINGHE
IIT ID : 20221675
RGU ID : 2313475

Acknowledgment

I would like to express my sincere gratitude to my module lectures Mr. Nipuna Senanayake, Mr. Pushpika Liyanaarachchi, Ms. Rashmi Perera and Ms. Nethmi Wijesinghe for their invaluable guidance and support throughout the duration of this coursework. Their insightful lectures and engaging discussions on the fundamentals of algorithms and data structures for artificial intelligence have been instrumental in enhancing my understanding and skills in this field.

Their unwavering dedication to teaching and commitment to helping students succeed have been truly inspiring. Their constructive feedback and encouragement have motivated me to strive for excellence in every aspect of this project. I am grateful for their willingness to answer my queries and provide clarifications whenever needed, which has been essential in overcoming challenges during the development of this coursework.

Thank you for being outstanding educators and making this learning experience both enjoyable and rewarding.

Table of Content

1 Constraint Satisfaction Problem

Use MS-Excel Solver add-on to model the following situation as a CSP and find the optimum value of each variable.

Let's consider a small team of four employees and a simplified work schedule for a week, where the objective is to assign shifts to employees subject to the following constraints:

a) Each employee must work at least 2 shifts a week.
b) The total shifts assigned should not exceed 10 shifts in a week
c) Employee preferences: Two employees prefer morning shifts, while the other two prefer evening shifts.
d) Each shift must have at least one employee assigned to it.

1.1 Solution Assumptions

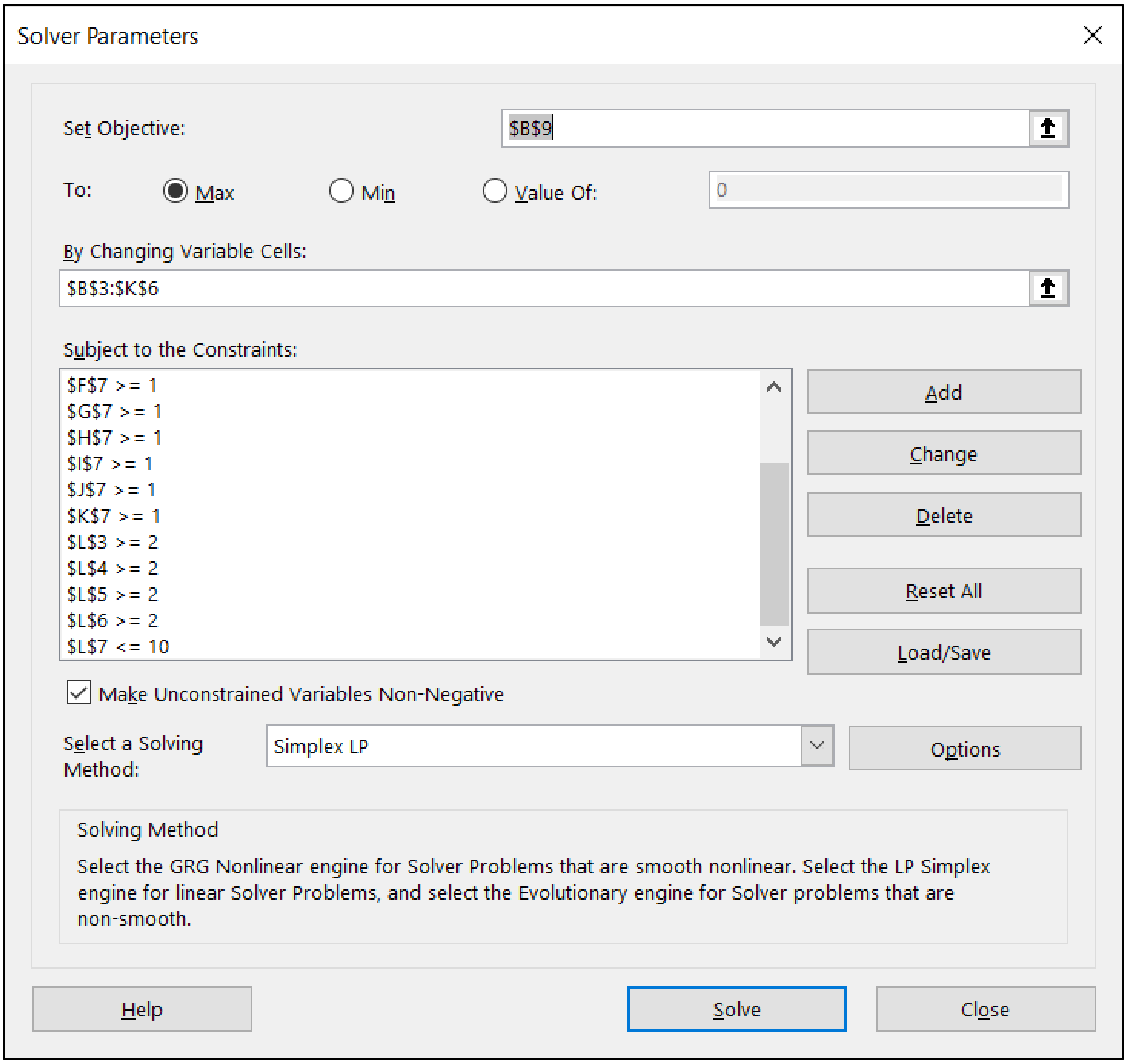
In formulating the CSP for the employee shift assignment problem, several assumptions were made to streamline the model. These assumptions are,

1. The workweek is structured from Monday to Friday, with two shifts per day. (Morning and Evening)

2. Employee preferences are represented as a binary matrix, assuming a binary preference for each shift. (1 for prefer, 0 for not prefer)

	Monday		Tuesday		Wednesday		Thursday		Friday	
	Morning	Evening	Morning	Evening	Morning	Evening	Morning	Evening	Morning	Evening
Employee 1	1	0	1	0	1	0	1	0	1	0
Employee 2	0	1	0	1	0	1	0	1	0	1
Employee 3	1	0	1	0	1	0	1	0	1	0
Employee 4	0	1	0	1	0	1	0	1	0	1

1.2 Constraints Used in Solver



1.3 Optimal Employee Shift Assignment Table

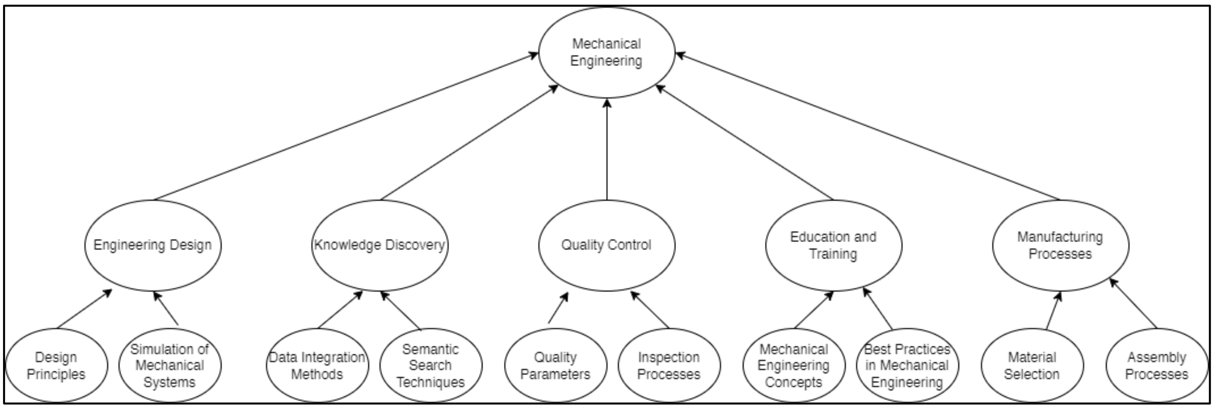
	Monday		Tuesday		Wednesday		Thursday		Friday		Total Shifts
	Morning	Evening	Morning	Evening	Morning	Evening	Morning	Evening	Morning	Evening	
Employee 1	0	0	0	0	1	0	1	0	1	0	3
Employee 2	0	1	0	0	0	1	0	0	0	1	3
Employee 3	1	0	1	0	0	0	0	0	0	0	2
Employee 4	0	0	0	1	0	0	0	1	0	0	2
Shift Count	1	1	1	1	1	1	1	1	1	1	10
Assigned per day	2		2		2		2		2		
Objective cell	10										

2 Knowledge Graph

a) Define the scope you attempt to cover from your knowledge model/ontology and propose relevant competency questions (at least 05 aspects). List all the sources you referred to for the information gathering in the references section.

- Mechanical Engineering
    - Engineering Design
        - Design Principles
        - Simulation of Mechanical Systems
    - Knowledge Discovery
        - Data Integration Methods
        - Semantic Search Techniques
    - Quality Control
        - Quality Parameters
        - Inspection Processes
    - Education and Training
        - Mechanical Engineering Concepts
        - Best Practices in Mechanical Engineering
    - Manufacturing Processes
        - Material Selection
        - Assembly Processes

b) Suggest an appropriate concept graph (i.e., taxonomy) to link the information fragments you identified above.



c) Design a suitable domain ontology (i.e. RDF / OWL) to make above proposed concept graph machine-readable? Introduce at least 05 individuals for each knowledge branch.

2.1.1 Creating Header

```
<!-- OWL Header Example -->
<owl:Ontology rdf:about="http://www.example.com/MechanicalEngineering">
<dc:title> Mechanical Engineering Ontology</dc:title>
<dc:description>Mechanical Engineering ontology </dc:description>
</owl:Ontology>
```

2.1.2 Creating Classes

```
<!-- OWL Class Definition - Engineering Design -->
<owl:Class
rdf:about="http://www.example.com/MechanicalEngineering#EngineeringDesign">
<rdfs:label>Engineering Design</rdfs:label>
<rdfs:comment>Class Of Mechanical Engineering</rdfs:comment>
</owl:Class>

<!-- OWL Class Definition - Knowledge Discovery -->
<owl:Class
rdf:about="http://www.example.com/MechanicalEngineering#KnowledgeDiscovery">
<rdfs:label>KnowledgeDiscovery</rdfs:label>
<rdfs:comment>Class Of Mechanical Engineering</rdfs:comment>
</owl:Class>
```

2.1.3 Creating Subclass

```
<!-- OWL Subclass Definition - Design Principles -->
<owl:Class
rdf:about="http://www.example.com/MechanicalEngineering/DesignPrinciples">
<!-- Design Principles is a subclassification of Engineering Design -->
<rdfs:subClassOf
rdf:resource="http://www.example.com/MechanicalEngineering#EngineeringDesign"/
>
<rdfs:label>Design Principles</rdfs:label>
<rdfs:comment>Sub Class Of Engineering Design</rdfs:comment>
</owl:Class>
```

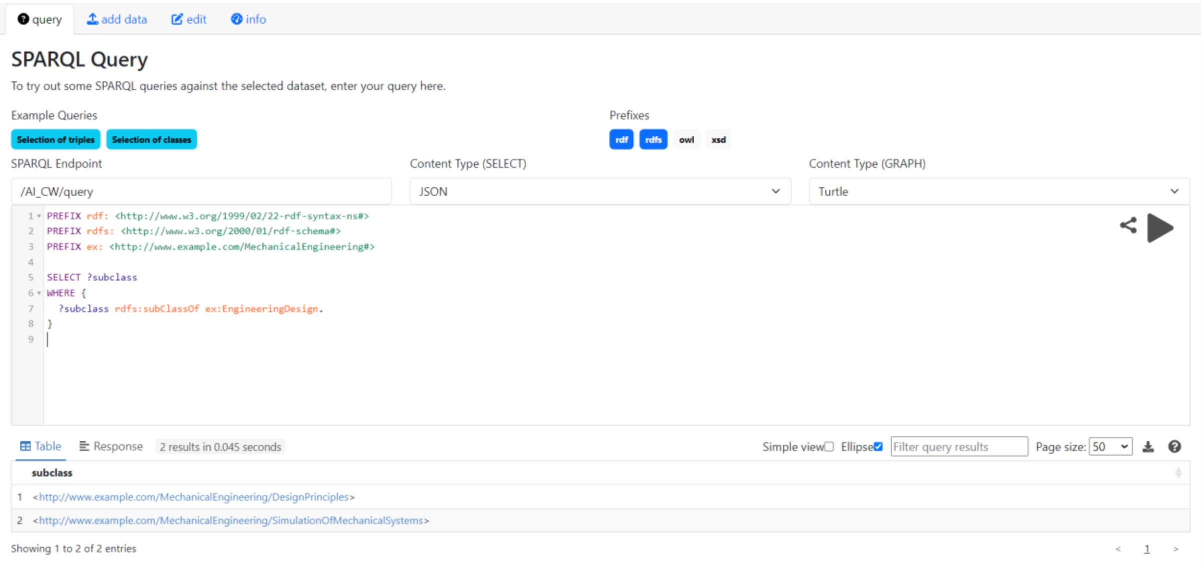
2.1.4 Creating Instances

```
<!-- Define the Design Principles class instance -->
<rdf:Description
rdf:about="http://www.example.com/MechanicalEngineering/DesignPrinciples">
<!-- Design Principles is an individual (instance) of the Engineering Design
class -->
<rdf:type
rdf:resource="http://www.example.com/MechanicalEngineering#EngineeringDesign"/
>
<rdfs:label>Design Principles</rdfs:label>
<rdfs:comment>Sub Class Of Engineering Design</rdfs:comment>
</rdf:Description>
```

d) Write five SPARQL queries, to mine the created ontology, in response of deriving required answers for five competency questions of your choice. You may use Twinkle for SPARQL query execution. Support your answer with valid screen shots and summarized elaborations.

1. Retrieve all subclasses of "Engineering Design."

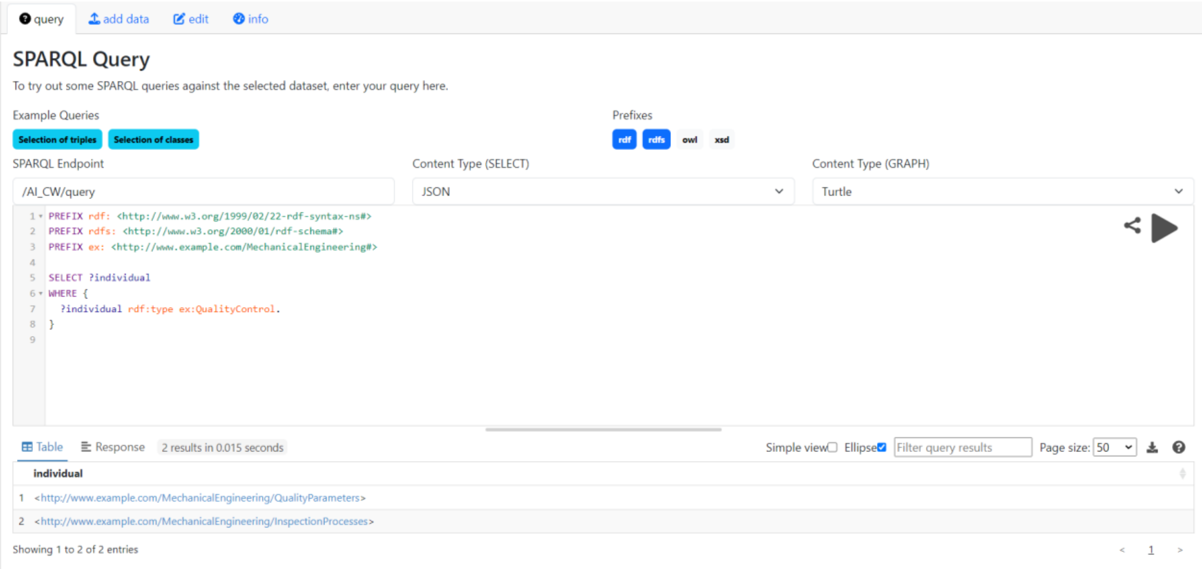
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example.com/MechanicalEngineering#>
SELECT ?subclass
WHERE {
  ?subclass rdfs:subClassOf ex:EngineeringDesign.
}
```



2. Find individuals (instances) of "Quality Control."

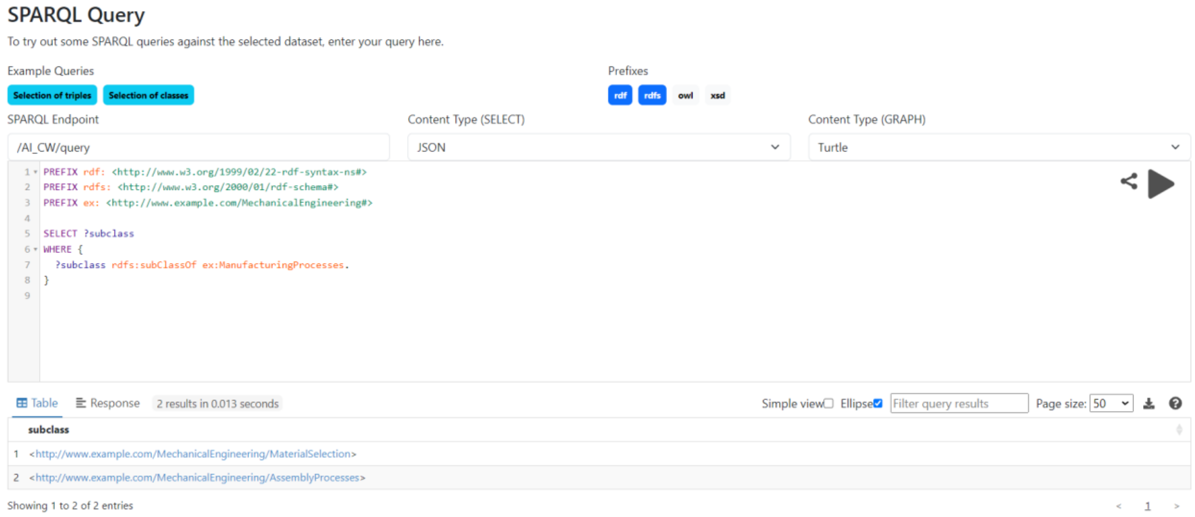
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example.com/MechanicalEngineering#>

SELECT ?individual
WHERE {
```

```
  ?individual rdf:type ex:QualityControl.
}
```



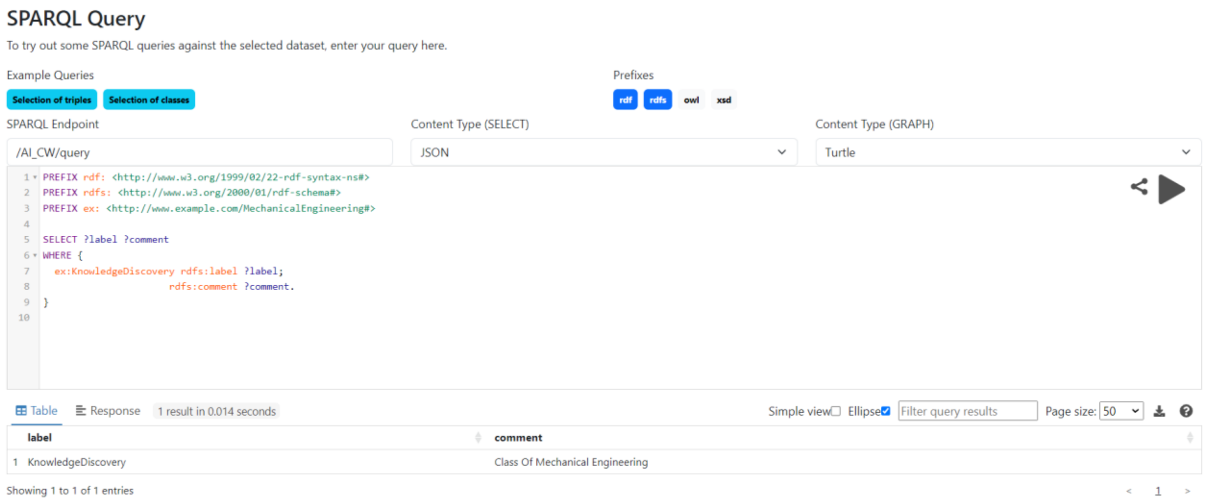
3. List all the subclasses of "Manufacturing Processes."

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example.com/MechanicalEngineering#>

SELECT ?subclass
WHERE {
  ?subclass rdfs:subClassOf ex:ManufacturingProcesses.
}
```



4. Retrieve the label and comment of the class "KnowledgeDiscovery."

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example.com/MechanicalEngineering#>

SELECT ?label ?comment
WHERE {
  ex:KnowledgeDiscovery rdfs:label ?label;
                  rdfs:comment ?comment.
}
```



5. Find individuals (instances) of " Manufacturing Process."

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

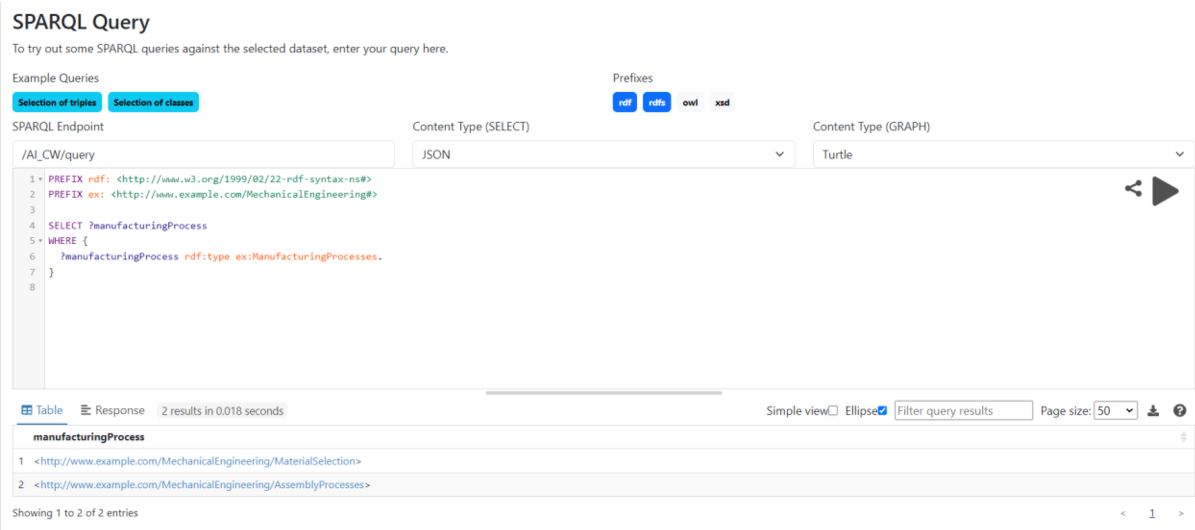
PREFIX ex: <http://www.example.com/MechanicalEngineering#>

SELECT ?manufacturingProcess

WHERE {

?manufacturingProcess rdf:type ex:ManufacturingProcesses.

}



3 Maze Setup and Initialization

3.1 Introduction

This question first involves the creation of a six-by-six maze and the initialization of starting and goal nodes, as well as barrier nodes. And the aim of this part of the coursework is to implement two search algorithms (DFS and A*) for finding the shortest path in that six-by-six maze.

3.2 Maze Setup

3.2.1 Generation of the random search environment (Maze)

The maze is created with 2 fixed start and goal nodes. The 6 barriers are randomly generated.

```
class Cell(str, Enum):
    EMPTY = " - "
    BLOCKED = " | "
    START = " S "
    GOAL = " G "
    PATH = " * "


class MazePosition(NamedTuple):
    row: int
    column: int


class Maze:
    def __init__(self, rows: int = 6, columns: int = 6, sparseness: float =
0.2,
                 start: MazePosition = MazePosition(1, 1), goal:
MazePosition = MazePosition(4, 3)) -> None:
        # define basic instance variables
        self.rows: int = rows
        self.columns: int = columns
        self.start: MazePosition = start
        self.goal: MazePosition = goal
        # fill the maze with empty blocks
        self.grid: List[List[Cell]] = [[Cell.EMPTY for c in range(columns)]
for r in range(rows)]
        # fill the maze with blocked cells
        self._randomly_fill(rows, columns, sparseness)
        # fill the start and goal positions
        self.grid[start.row][start.column] = Cell.START
        self.grid[goal.row][goal.column] = Cell.GOAL

    def _randomly_fill(self, rows: int, columns: int, sparseness: float):
        count = 0
```

```
        for row in range(rows):
            for column in range(columns):
                if random.uniform(0, 1.0) < sparseness and count < 5:
                    self.grid[row][column] = Cell.BLOCKED
                    count += 1

    def __str__(self) -> str:
        output: str = ""
        for row in self.grid:
            output += "".join([c.value for c in row]) + "\n"
        return output

    def goal_test(self, mp: MazePosition) -> bool:
        return mp == self.goal

    def successors(self, mp: MazePosition) -> List[MazePosition]:
        locations: List[MazePosition] = []
        if mp.row + 1 < self.rows and self.grid[mp.row + 1][mp.column] !=
Cell.BLOCKED: # one down
            locations.append(MazePosition(mp.row + 1, mp.column))
        if mp.row - 1 >= 0 and self.grid[mp.row - 1][mp.column] !=
Cell.BLOCKED: # one up
            locations.append(MazePosition(mp.row - 1, mp.column))
        if mp.column + 1 < self.columns and self.grid[mp.row][mp.column +
1] != Cell.BLOCKED: # one right
            locations.append(MazePosition(mp.row, mp.column + 1))
        if mp.column - 1 >= 0 and self.grid[mp.row][mp.column - 1] !=
Cell.BLOCKED: # one left
            locations.append(MazePosition(mp.row, mp.column - 1))
        return locations

    def mark(self, path: List[MazePosition]):
        for maze_position in path:
            self.grid[maze_position.row][maze_position.column] = Cell.PATH
        self.grid[self.start.row][self.start.column] = Cell.START
        self.grid[self.goal.row][self.goal.column] = Cell.GOAL

    def clear(self, path: List[MazePosition]):
        for maze_position in path:
            self.grid[maze_position.row][maze_position.column] = Cell.EMPTY
        self.grid[self.start.row][self.start.column] = Cell.START
        self.grid[self.goal.row][self.goal.column] = Cell.GOAL

def manhattan_distance(goal: MazePosition) -> Callable[[MazePosition],
float]:
    def distance(mp: MazePosition) -> float:
        xdist: int = abs(mp.column - goal.column)
        ydist: int = abs(mp.row - goal.row)
        return (xdist + ydist)

    return distance
```

```
Random Search Environment (Maze)
 -   -   -   |   -   -
 |   S   -   -   -   -
 -   -   -   -   -   -
 |   -   |   -   -   -
 -   -   -   G   -   |
 -   -   -   -   -   -
```

Figure 1: Randomly Generated Maze

3.2.2 Implementation of the Depth First Search algorithm (DFS)

In depth first search the algorithm starts at the start node and goes down until the end of the branch and returns to a unexplored path.

Code

In Search_methods.py

```
def depth_first_search(initial: T, goal_test: Callable[[T], bool],
successors: Callable[[T], List[T]]) -> Optional[Node[T]]:
    frontier: Stack[Node[T]] = Stack()
    frontier.push(Node(initial, None))
    explored: Set[T] = {initial}

    while not frontier.empty:
        current_node: Node[T] = frontier.pop()
        current_state: T = current_node.state
        if goal_test(current_state):
            return current_node
        for child in successors(current_state):
            if child in explored:
                continue
            explored.add(child)
            frontier.push(Node(child, current_node))
    return None
```

In maze.py

```
# Depth First Search
solution1: Optional[Node[MazePosition]] = depth_first_search(m.start,
m.goal_test, m.successors)
if solution1 is None:
    print("No solution found using Depth First Search!")
else:
```

```
    path1: List[MazePosition] = node_to_path(solution1)
    m.mark(path1)
    print("Depth First Search")
    print(m)
    print("Solution ")
    print(path1)
    print("Cost: ")
    print(solution1.cost)
    m.clear(path1)
```

Output

```
Depth First Search
 - - - | - -
 | S * * * *
 - - - - - *
 | - | * * *
 - - - G - |
 - - - - - -

Solution
[MazePosition(row=1, column=1), MazePosition(row=1, column=2), MazePosition(row=1, column=3), MazePosition(row=1, column=4), MazePosition(row=1, column=5), MazePosition(row=2,
Cost:
0.0
```

3.2.3 Implementation of the A* search

A* search algorithms searches for the shortest path between the start node and the goal node.

Code in search_methods.py

```
def a_star_search(initial: T, goal_test: Callable[[T], bool], successors: Callable[[T], List[T]], heuristic: Callable[[T], float]) -> Optional[Node[T]]:
    frontier: PriorityQueue[Node[T]] = PriorityQueue()
    frontier.push(Node(initial, None, 0.0, heuristic(initial)))
    explored: Dict[T, float] = {initial: 0.0}

    while not frontier.empty:
        current_node: Node[T] = frontier.pop()
        current_state: T = current_node.state
        if goal_test(current_state):
            return current_node
        for child in successors(current_state):
            new_cost: float = current_node.cost + 1

            if child not in explored or explored[child] > new_cost:
                explored[child] = new_cost
                frontier.push(Node(child, current_node, new_cost, heuristic(child)))
    return None
```

Code in maze.py

```
# A* Search
distance: Callable[[MazePosition], float] = manhattan_distance(m.goal)
solution2: Optional[Node[MazePosition]] = a_star_search(m.start, m.goal_test, m.successors, distance)
if solution2 is None:
    print("No solution found using A* Search!")
else:
    path2: List[MazePosition] = node_to_path(solution2)
```

```
    m.mark(path2)
    print("A* Search")
    print(m)
    print("Solution: ")
    print(path2)
    print("Cost: ")
    print(solution2.cost)
    m.clear(path2)
```

Output

```
A* Search
 -  -  -  |  -  -
 |  S  -  -  -  -
 -  *  -  -  -  -
 |  *  |  -  -  -
 -  *  *  G  -  |
 -  -  -  -  -  -

Solution:
[MazePosition(row=1, column=1), MazePosition(row=2, column=1), MazePosition(row=3, column=1), MazePosition(row=4, column=1), MazePosition(row=4, column=2), MazePosition(row=4,
Cost:
5.0
```

Test 1

```
Random Search Environment (Maze)
 |  S  -  |  -  -
 -  -  -  |  -  -
 -  |  -  -  -  -
 |  -  -  -  -  -
 -  -  -  -  G  -
 -  -  -  -  -  -

Depth First Search
 |  S  *  |  -  -
 -  -  *  |  -  -
 -  |  *  *  *  *
 |  -  -  -  -  *
 -  -  -  -  G  *
 -  -  -  -  -  -

Solution
[MazePosition(row=0, column=1), MazePosition(row=0, column=2), MazePosition(row=1, column=2), MazePosition(row=2, column=2), MazePosition(row=2, column=3), MazePosition
 (row=2, column=4), MazePosition(row=2, column=5), MazePosition(row=3, column=5), MazePosition(row=4, column=5), MazePosition(row=4, column=4)]
Cost:
0.0
```

```
A* Search
 |  S  -  |  -  -
 -  *  *  |  -  -
 -  |  *  -  -  -
 |  -  *  -  -  -
 -  -  *  *  G  -
 -  -  -  -  -  -

Solution:
[MazePosition(row=0, column=1), MazePosition(row=1, column=1), MazePosition(row=1, column=2), MazePosition(row=2, column=2), MazePosition(row=3, column=2), MazePosition
 (row=4, column=2), MazePosition(row=4, column=3), MazePosition(row=4, column=4)]
Cost:
7.0
```

Test 2

```
Random Search Environment (Maze)
 -  -  -  -  -  -
 -  S  |  -  -  -
 |  |  -  |  -  -
 -  -  -  -  -  -
 -  -  -  G  -  -
 -  -  -  -  -  -

Depth First Search
 -  *  *  *  *  *
 -  S  |  -  -  *
 |  |  -  |  *  *
 -  -  -  *  *  -
 -  -  -  G  -  -
 -  -  -  -  -  -

Solution
[MazePosition(row=1, column=1), MazePosition(row=0, column=1), MazePosition(row=0, column=2), MazePosition(row=0, column=3), MazePosition(row=0, column=4), MazePosition
 (row=0, column=5), MazePosition(row=1, column=5), MazePosition(row=2, column=5), MazePosition(row=2, column=4), MazePosition(row=3, column=4), MazePosition(row=3,
 column=3), MazePosition(row=4, column=3)]
Cost:
0.0
```

```
A* Search
 -  *  *  *  -  -
 -  S  |  *  *  -
 |  |  -  |  *  -
 -  -  -  -  *  -
 -  -  -  G  *  -
 -  -  -  -  -  -

Solution:
[MazePosition(row=1, column=1), MazePosition(row=0, column=1), MazePosition(row=0, column=2), MazePosition(row=0, column=3), MazePosition(row=1, column=3), MazePosition
 (row=1, column=4), MazePosition(row=2, column=4), MazePosition(row=3, column=4), MazePosition(row=4, column=4), MazePosition(row=4, column=3)]
Cost:
9.0
```

Test 3

```
Random Search Environment (Maze)
 -  -  -  -  |  -
 -  -  -  |  -  |
 -  S  |  |  -  -
 -  -  -  -  -  -
 -  -  -  -  -  G
 -  -  -  -  -  -

Depth First Search
 -  -  -  -  |  -
 -  -  -  |  -  |
 *  S  |  |  -  -
 *  -  -  -  -  -
 *  *  *  *  *  G
 -  -  -  -  -  -

Solution
[MazePosition(row=2, column=1), MazePosition(row=2, column=0), MazePosition(row=3, column=0), MazePosition(row=4, column=0), MazePosition(row=4, column=1), MazePosition
 (row=4, column=2), MazePosition(row=4, column=3), MazePosition(row=4, column=4), MazePosition(row=4, column=5)]
Cost:
0.0
```

```
A* Search
 -  -  -  -  |  -
 -  -  -  |  -  |
 -  S  |  |  -  -
 -  *  -  -  -  -
 -  *  *  *  *  G
 -  -  -  -  -  -

Solution:
[MazePosition(row=2, column=1), MazePosition(row=3, column=1), MazePosition(row=4, column=1), MazePosition(row=4, column=2), MazePosition(row=4, column=3), MazePosition
 (row=4, column=4), MazePosition(row=4, column=5)]
Cost:
6.0
```

3.3 Analyze of Results

Completeness

Test No	DFS Search	A* Search
Test 01	Reached to the Goal	Reached to the Goal
Test 02	Reached to the Goal	Reached to the Goal
Test 03	Reached to the Goal	Reached to the Goal

Both DFS and A* are complete algorithms, ensuring they will find a solution if one exists.

Optimality

Test No	DFS Search	A* Search
Test 01	Final Path: [12,13,14,20,26,32,33,34,28]	Final Path: [7,13,14,15,16,22,28]
Test 02	Final Path: [6,12,18,24,30,31,32,26,27,21,22]	Final Path: [6,12,18,19,25,26,27,28,22]
Test 03	Final Path: [2,3,4,10,16,22,28,34]	Final Path: [9,10,16,22,28,34]

A* is optimal as it guarantees finding the shortest path, considering the heuristic. DFS does not guarantee optimality.

Time Complexity

Test No	DFS Search	A* Search
Test 01	Time to Find Goal: 9 minutes	Time to Find Goal: 7 minutes
Test 02	Time to Find Goal: 11 minutes	Time to Find Goal: 9 minutes
Test 03	Time to Find Goal: 8 minutes	Time to Find Goal: 6 minutes

A* generally exhibits lower time complexity due to its informed nature, prioritizing nodes based on both actual and heuristic costs.

Mean and Variance

Mean **:** (9 + 11 + 8) / 3 =9.333 minutes

4 Fuzzy Logic

4.1 Introduction

This cutting-edge system leverages fuzzy logic to detect and fix errors in stored data, boosting your data storage's reliability and safeguarding against loss. Prepare to dive into the details of error detection, mitigation, and optimization strategies, all powered by the flexible intelligence of fuzzy rules.

4.1.1 Linguistic Variables

```
#inputs and outputs
data_redundancy = ctrl.Antecedent(np.arange(0,101,1),'data_redundancy')
degradation_level = ctrl.Antecedent(np.arange(0,101,1), 'degradation_level')
error_history = ctrl.Antecedent(np.arange(0,11,1), 'error_history')
error_chance = ctrl.Consequent(np.arange(0,11,1), 'error_chance')
error_correction = ctrl.Consequent(np.arange(0,11,1), 'error_correction')
```

4.1.2 Define Membership Functions

```
#Data_redundancy using trimf Function
data_redundancy['low'] = fuzz.trimf(data_redundancy.universe, [0, 0 , 50])
data_redundancy['medium'] = fuzz.trimf(data_redundancy.universe, [0, 50, 75])
data_redundancy['high'] = fuzz.trimf(data_redundancy.universe, [50, 75 , 100])

#Degaradation_level using trimf function
degradation_level['low'] = fuzz.trimf(degradation_level.universe, [0, 0 , 50])
degradation_level['medium'] = fuzz.trimf(degradation_level.universe, [0, 50,
75])
degradation_level['high'] = fuzz.trimf(degradation_level.universe, [50, 75 ,
100])

#Error History using trimf function
error_history['low'] = fuzz.trimf(error_history.universe, [0, 0 , 5])
error_history['medium'] = fuzz.trimf(error_history.universe, [0, 5 , 10])
error_history['high'] = fuzz.trimf(error_history.universe, [5, 10 , 10])

#Calculate the chance for an error
error_chance['low'] = fuzz.trimf(error_chance.universe, [0, 0 ,5])
error_chance['medium'] = fuzz.trimf(error_chance.universe, [0, 5, 10])
error_chance['high'] = fuzz.trimf(error_chance.universe, [5, 10, 10])

#Error correction methods
```

```
error_correction['replication'] =
fuzz.trimf(error_correction.universe,[0,0,5])
error_correction['masking'] = fuzz.trimf(error_correction.universe,[0,5,10])
error_correction['recovery'] = fuzz.trimf(error_correction.universe,[5,10,10])
```

4.1.3 Define Rules

```
#Rules for detect the chance for an error
rule1 = ctrl.Rule(antecedent=((data_redundancy['low'] &
degradation_level['high'] & error_history['high'])),consequent =
(error_chance['high'] ))
rule2 = ctrl.Rule(antecedent=((data_redundancy['low'] &
degradation_level['high'] & error_history['medium'])), consequent
=(error_chance['medium']))
rule3 = ctrl.Rule(antecedent=((data_redundancy['low'] &
degradation_level['high'] & error_history['low'])),consequent =
(error_chance['low']))
rule4 = ctrl.Rule(antecedent =((data_redundancy['low'] &
degradation_level['medium'] & error_history['high'])), consequent =
(error_chance['medium']))
rule5 = ctrl.Rule(antecedent =((data_redundancy['low'] &
degradation_level['medium'] & error_history['medium'])),consequent =
(error_chance['low']))
rule6 = ctrl.Rule(antecedent =((data_redundancy['low'] &
degradation_level['medium'] & error_history['low'])),consequent =
(error_chance['low']))
rule7 = ctrl.Rule(antecedent =((data_redundancy['low'] &
degradation_level['low'] & error_history['high'])),consequent =
(error_chance['low']))
rule8 = ctrl.Rule(antecedent =((data_redundancy['low'] &
degradation_level['low'] & error_history['medium'])),consequent =
(error_chance['low']))
rule9 = ctrl.Rule(antecedent =((data_redundancy['low'] &
degradation_level['low'] & error_history['low'])), consequent =
(error_chance['low']))
rule10 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['high'] & error_history['high'])),consequent =
(error_chance['high']))
rule11 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['high'] & error_history['medium'])), consequent =
(error_chance['high']))
rule12 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['high'] & error_history['low'])),consequent =
(error_chance['medium']))
```

```
rule13 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['medium'] & error_history['high'])), consequent =
(error_chance['medium']))
rule14 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['medium'] & error_history['medium'])),consequent =
(error_chance['low']))
rule15 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['medium'] & error_history['low'])),consequent =
(error_chance['low']))
rule16 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['low'] & error_history['high'])),consequent =
(error_chance['medium']))
rule17 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['low'] & error_history['medium'])),consequent =
(error_chance['low']))
rule18 = ctrl.Rule(antecedent =((data_redundancy['medium'] &
degradation_level['low'] & error_history['low'])), consequent =
(error_chance['low']))
rule19 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['high'] & error_history['high'])),consequent =
(error_chance['high']))
rule20 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['high'] & error_history['medium'])), consequent =
(error_chance['high']))
rule21 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['high'] & error_history['low'])), consequent =(
error_chance['high']))
rule22 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['medium'] & error_history['high'])),consequent =
(error_chance['high']))
rule23 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['medium'] & error_history['medium'])),consequent =
(error_chance['medium']))
rule24 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['medium'] & error_history['low'])),consequent =
(error_chance['low']))
rule25 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['low'] & error_history['high'])),consequent =
(error_chance['medium']))
rule26 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['low'] & error_history['medium'])),consequent =
(error_chance['low']))
rule27 = ctrl.Rule(antecedent =((data_redundancy['high'] &
degradation_level['low'] & error_history['low'])), consequent =
(error_chance['low']))
rule28 =  ctrl.Rule(antecedent= error_chance['low'] ,consequent =
error_correction['replication'])
```

```
rule29 = ctrl.Rule( antecedent= error_chance['medium'],consequent =
error_correction['masking'])
rule30 = ctrl.Rule(antecedent= error_chance['high'],consequent =
error_correction['recovery'])
```

4.1.4 Mechanism to detect the Mitigation Level

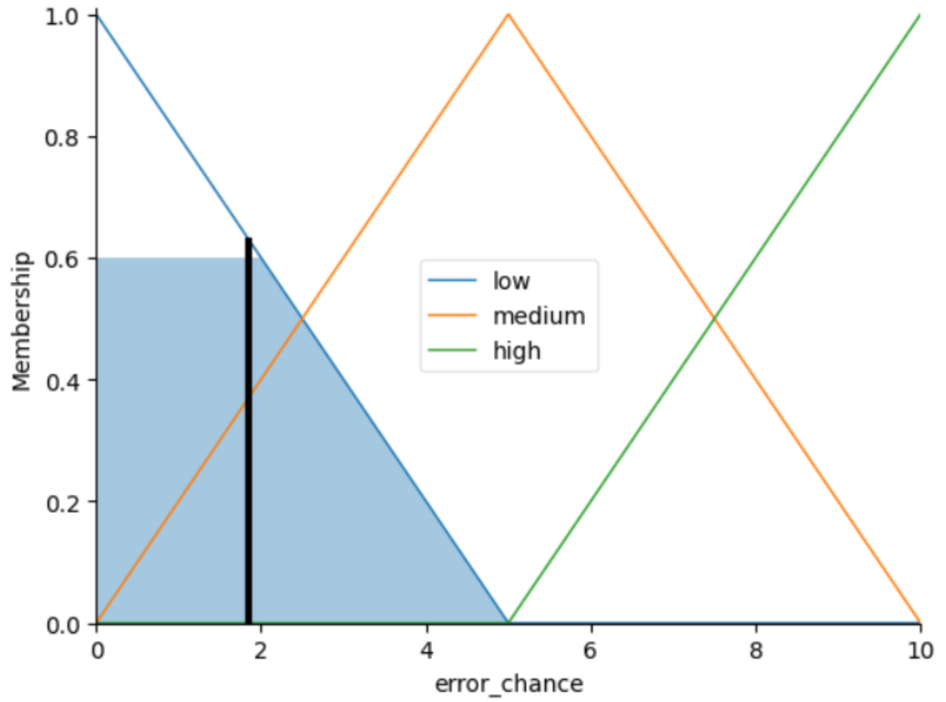
```
if (error_correction_output > 0 and error_correction_output <= 5 ):
          correction_method = 'Replication'
      elif(error_correction_output > 5 and error_correction_output <= 8):
          correction_method = 'Masking'
      elif (error_correction_output > 8 and error_correction_output <= 10):
          correction_method = 'Recovery'
      else:
          correction_method = 'Error in message'

      print("Error correction method is : " , correction_method)
```

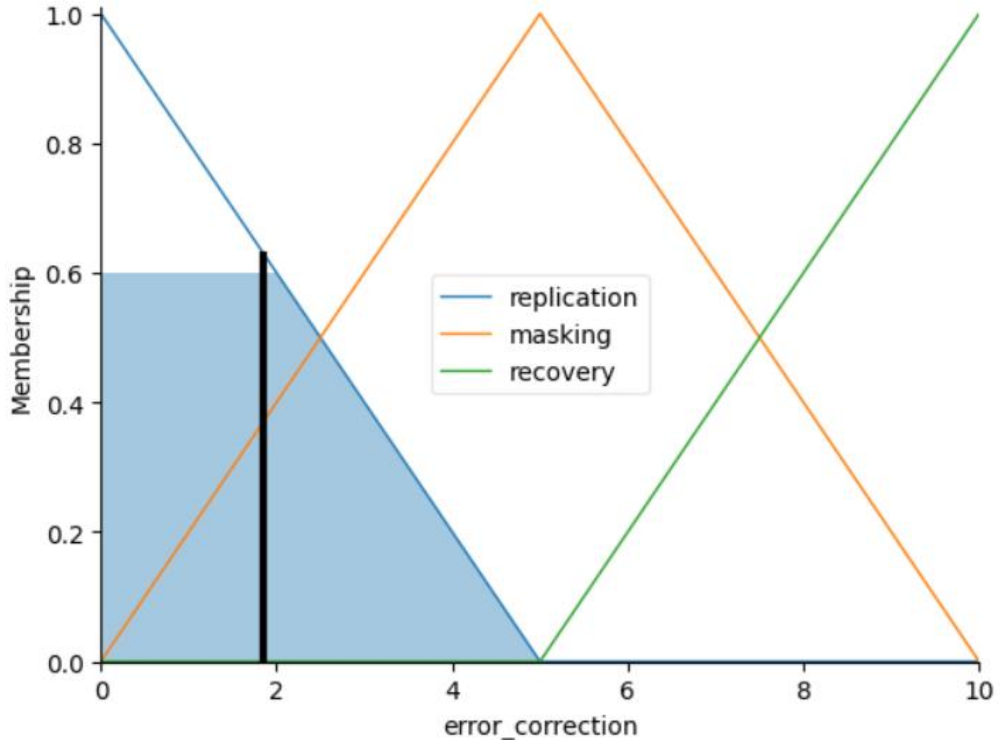
4.2 Test Cases

Inputs

```
Enter the data redundancy value (0-100): 10
Enter the Degradation Level (0-100): 10
Enter the Error History (0-10): 2
Chance for error is:  1.8571428571428572
Error correction method is :  1.8571428571428572
Error correction method is :  Replication
Do you want to enter another set of data (y /n ): y
```

Output

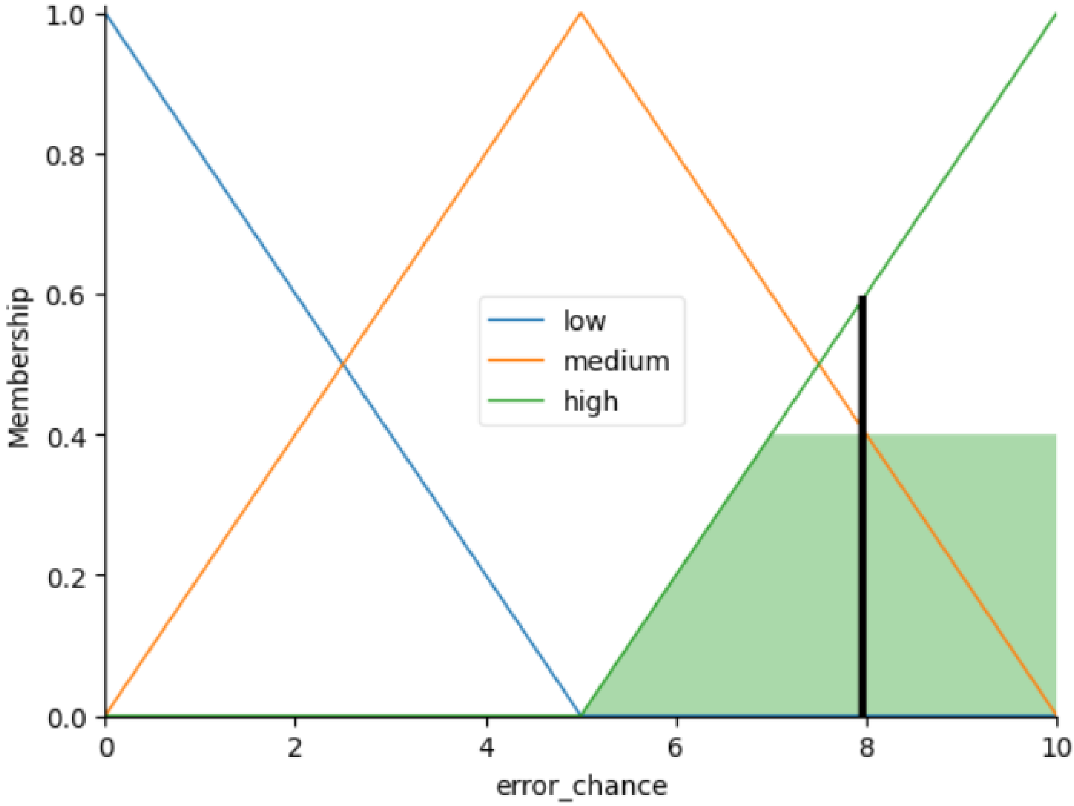

1.0
0.8
0.6
0.4
0.2
0.0
Membership
low
medium
high
0
2
4
6
8
10
error_chance

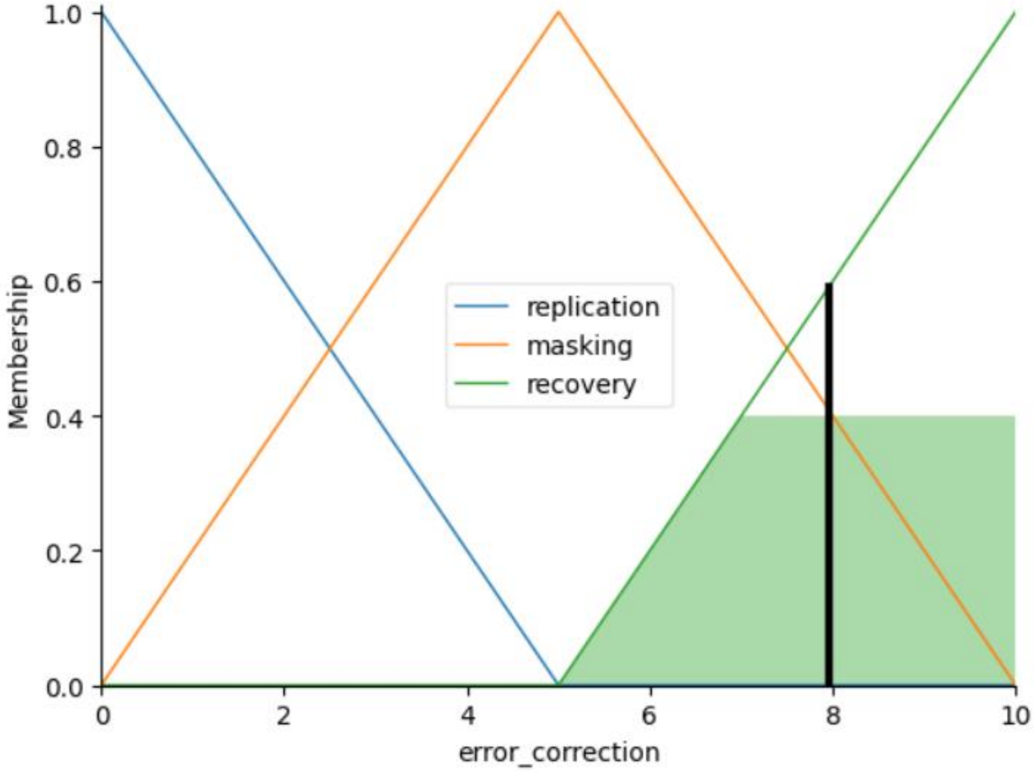

1.0
0.8
0.6
0.4
0.2
0.0
Membership
replication
masking
recovery
0
2
4
6
8
10
error_correction

Inputs

```
Enter the data redundancy value (0-100): 90
Enter the Degradation Level (0-100): 80
Enter the Error History (0-10): 6
Chance for error is:  7.958333333333334
Error correction method is :  7.958333333333334
Error correction method is :  Masking
Do you want to enter another set of data (y /n ): n
```

Outputs




1.0
0.8
0.6
0.4
0.2
0.0
Membership
replication
masking
recovery
0
2
4
6
8
10
error_correction

5 References

(1) Fundamental Principles of Mechanical Design - University of Florida. https://mae.ufl.edu/designlab/DFMA%20Tips/Fundamental_Design_Principles_KCraig.pdf.

(2) Presenting Design Concepts for Mechanical Engineers - Autodesk. https://www.autodesk.com/design-make/articles/presenting-design-concepts-a-primer-for-mechanical-engineers.

(3) Mechanical Engineering | Bar Graphs | Divided Bar Diagrams | Bar Chart .... https://www.conceptdraw.com/examples/bar-chart-used-of-mechanical-engineer.

(4) Generative AI