Ethan Fox

AERO 689 – Aerospace Autonomy

Project Submission

1. Plot/draw the Bayesian network and implement it using the code provided.

A diagram of a diagram

Description automatically generated

1. Use the code to answer the following test queries:

from project\_utils import \*

T, F = True, False

rover\_faults = BayesNet([

    # Base probabilities for each fault

    ('BatterySystemFault', '', 0.001),

    ('SolarPanelDust', '', 0.05),

    ('RocksInWheels', '', 0.02),

    # Conditional probabilities for symptoms given faults

    ('LowPower', 'BatterySystemFault SolarPanelDust',

        {(T, T): 0.99, (T, F): 0.95, (F, T): 0.75, (F, F): 0.01}),

    ('WheelSubsystemFailure', 'RocksInWheels',

        {(T): 0.9, (F): 0.05}),

    ('SlowMovement', 'LowPower WheelSubsystemFailure',

        {(T, T): 0.95, (T, F): 0.8, (F, T): 0.5, (F, F): 0.05}),

    ('WeakCommsSignal', 'LowPower',

        {(T): 0.8, (F): 0.3}),

    ('MovementDrift', 'WheelSubsystemFailure',

        {(T): 0.8, (F): 0.1})

])

# Part A

print("The probability of solar panels covered in dust given slow movement, enumeration method:",enumeration\_ask('SolarPanelDust', {'SlowMovement': T}, rover\_faults).show\_approx())

print("The probability of solar panels covered in dust given slow movement, elimination method:",elimination\_ask('SolarPanelDust', {'SlowMovement': T}, rover\_faults).show\_approx())

# Part B

print("The probability of a battery system fault given weak comms, enumeration method:",enumeration\_ask('BatterySystemFault', {'WeakCommsSignal': T}, rover\_faults).show\_approx())

print("The probability of a battery system fault given weak comms, elimination method:",elimination\_ask('BatterySystemFault', {'WeakCommsSignal': T}, rover\_faults).show\_approx())

# Part C

print("The probability of rock stuck in wheel given slow movement and low battery power, enumeration method:",enumeration\_ask('WheelSubsystemFailure', {'MovementDrift': T, 'LowPower': T }, rover\_faults).show\_approx())

print("The probability of rock stuck in wheel given slow movement and low battery power, elimination method:",elimination\_ask('WheelSubsystemFailure', {'MovementDrift': T, 'LowPower': T }, rover\_faults).show\_approx())

A black screen with white text

Description automatically generated

1. Integrate this code into the agent’s code.
   1. Complete the generate\_fault\_detection\_bayes\_net function.

def generate\_fault\_detection\_bayes\_net():

    # This function constructs a Bayesian network for fault detection using the provided PMFs.

    ### YOUR CODE HERE ### FOXY

    T, F = True, False

    fault\_detection\_bn = BayesNet([

        # Base probabilities for each fault

        ('BatteryFault', '', 0.001),

        ('SolarPanelDusty', '', 0.05),

        ('RockInWheel', '', 0.02),

        # Conditional probabilities for symptoms given faults

        ('lowBattery', 'BatteryFault SolarPanelDusty',

            {(T, T): 0.99, (T, F): 0.95, (F, T): 0.75, (F, F): 0.01}),

        ('wheelFailure', 'RockInWheel',

            {(T): 0.9, (F): 0.05}),

        ('SlowMovement', 'lowBattery wheelFailure',

            {(T, T): 0.95, (T, F): 0.8, (F, T): 0.5, (F, F): 0.05}),

        ('WeakComms', 'lowBattery',

            {(T): 0.8, (F): 0.3}),

        ('MovementDrifts', 'wheelFailure',

            {(T): 0.8, (F): 0.1})

    ])

    ### YOUR CODE HERE ###

    return fault\_detection\_bn

1. Complete the get\_fault\_prediction function.

def get\_fault\_prediction(fault\_info, fault\_bn):

    # This function predicts the fault state for the 3 different fault root causes (BatteryFault, SolarPanelDusty, RockInWheel) and puts them in a dict.

    prediction\_dict = {

        "BatteryFault": 0.0,

        "SolarPanelDusty": 0.0,

        "RockInWheel": 0.0

    }

    ### YOUR CODE HERE ### FOXY

    # Ask the fault\_bn bayes net for the probability of each fault conditioned on the fault\_info.

        # Convert fault\_info into a dictionary format suitable for Bayesian queries

    evidence = {}

    for symptom in fault\_info:

        evidence[symptom] = True  # Assuming all symptoms in fault\_info are observed as True

    # Querying the Bayesian network for each fault

    for fault in prediction\_dict.keys():

        query\_result = elimination\_ask(fault, evidence, fault\_bn)

        # Store the probability of the fault being True

        prediction\_dict[fault] = query\_result[True]

    ### YOUR CODE HERE ###

    return prediction\_dict

1. Report the % of wins with fault prediction vs without fault prediction capabilities. You can do this with Problem 2 complete or not, but please make clear which is the case, as you will have different victory %s.

Problem 2 is not complete; the results are as follows:

* With fault detection (ran 300 times):
  + 41%
* Without Fault Detection (300 times):
  + 35%

1. [Grad students only]: Find a way to improve upon the get\_fault\_reaction function. Show that your solution is better than the current get\_fault\_reaction function, by shortening the elapsed sim time or increasing the number of victories.

def get\_fault\_reaction(fault\_prediction):

    # Define thresholds for reacting to faults

    reaction\_thresholds = {

        "BatteryFault": 0.09,        # Threshold for reacting to BatteryFault

        "SolarPanelDusty": 0.4,     # Threshold for reacting to SolarPanelDusty

        "RockInWheel": 0.09         # Threshold for reacting to RockInWheel

    }

    # Initialize the fault reaction list

    fault\_reaction = []

    # Iterate through each fault and check if it exceeds the reaction threshold

    for fault, probability in fault\_prediction.items():

        # Check if the fault is in the reaction\_thresholds dictionary

        if fault in reaction\_thresholds:

            # Compare the probability of the fault with its threshold

            if probability > reaction\_thresholds[fault]:

                # If the probability exceeds the threshold, add the fault to the reaction list

                fault\_reaction.append(fault)

    return fault\_reaction

The original `get\_fault\_reaction` function uses hard-coded conditional statements to check each fault's probability individually and directly specifies threshold values and actions for each fault within the function. This approach lacks flexibility and scalability, as it requires manual modifications to the code for adjusting thresholds or adding new faults and corresponding actions.

In contrast, the updated `get\_fault\_reaction` function, with the introduction of the `reaction\_thresholds` dictionary, dynamically checks each fault against its threshold within a loop. This design promotes flexibility and maintainability, as it allows you to easily modify thresholds or add new faults and actions by updating the `reaction\_thresholds` dictionary without altering the core logic of the function. This approach enhances code readability and reduces redundancy, providing a more adaptable and centralized solution for reacting to faults based on their predicted probabilities. I was able to decrease sim time, from an average of 192.73 (over 1000 runs) to 186.27 (over 1000 runs). I understand the difference Is almost negligible, but I could not determine a better way to improve the policy.

**Problem 2**

Your task is to implement the probabilistic reasoning portion of the code, which will help reduce the number of rovers lost to storms. That is, when exploring, you should use your storm probabilities to help you break ties between possible actions.

1. Completion of the compute\_storm\_probs function for Problem 2

def compute\_storm\_probs(bayes\_nets, ppm\_evidence, sandyground\_evidence, facts, grid):

    prob\_grid = np.zeros((6, 6))

    # Process facts to create a map of cell properties

    cell\_properties = {(fact['xloc'], fact['yloc']): dict(fact) for fact in facts if hasattr(fact, 'template') and fact.template.name == 'cell'}

    # Combine ppm and sandyground evidence

    evidence\_dict = {}

    for key in ppm\_evidence:

        evidence\_dict[key] = {'PPM': ppm\_evidence[key]}

    for key in sandyground\_evidence:

        if key in evidence\_dict:

            evidence\_dict[key]['sandy'] = sandyground\_evidence[key]

        else:

            evidence\_dict[key] = {'sandy': sandyground\_evidence[key]}

    for i in range(6):

        for j in range(6):

            if cell\_properties.get((i, j), {}).get('safe') != 'true':

                combined\_evidence = {}

                # Add evidence for the current cell

                current\_ppm\_evidence = ppm\_evidence.get((i, j), {})

                current\_sandy\_evidence = sandyground\_evidence.get((i, j), {})

                if isinstance(current\_ppm\_evidence, dict):

                    combined\_evidence.update(current\_ppm\_evidence)

                if isinstance(current\_sandy\_evidence, dict):

                    combined\_evidence.update(current\_sandy\_evidence)

                # Iterate over neighboring cells

                for di in [-1, 0, 1]:

                    for dj in [-1, 0, 1]:

                        if di == 0 and dj == 0:

                            continue  # Skip the current cell

                        adj\_i, adj\_j = i + di, j + dj

                        if 0 <= adj\_i < 6 and 0 <= adj\_j < 6:

                            if cell\_properties.get((adj\_i, adj\_j), {}).get('safe') != 'true':

                                neighbor\_evidence = evidence\_dict.get((adj\_i, adj\_j), {})

                                if isinstance(neighbor\_evidence, dict):

                                    combined\_evidence.update(neighbor\_evidence)

                                # Calculate storm probability

                                storm\_relation = check\_storm\_nearby(grid, j, i)

                                storm\_prob = enumeration\_ask('StormProximity', combined\_evidence, bayes\_nets[(i, j)])

                                if storm\_relation in ['adjacent', 'diagonal']:

                                    prob\_grid[i][j] = storm\_prob[storm\_relation.capitalize()]

                                else:

                                    prob\_grid[i][j] = storm\_prob['NotNearby']

    return prob\_grid

1. Completion of the generate\_bayes\_nets function for Problem 2.

def generate\_bayes\_nets(nrows,ncols,storm\_prob,sandy\_prob):

    total\_cells = nrows \* ncols

    sandy\_prob = 3 / total\_cells  # Probability of a cell being sandy

    storm\_prob = [0.9, 0.25, 0.01]  # Probability of a storm

    bayesnets = {}

    for i in range(nrows):

        for j in range(ncols):

            # Conditional probability table (CPT) for 'PPM'

            # The CPT is based on the table provided

            ppm\_cpt = {

                ('Adjacent', True): [0.95,0.04,0.01],

                ('Adjacent', False): [0.9,0.09,0.01],

                ('Diagonal', True): [0.3,0.5,0.2],

                ('Diagonal', False): [0.25,0.5,0.25],

                ('NotNearby', True): [0.01,0.24,0.75],

                ('NotNearby', False): [0.01,0.09,0.9]

            }

            # Node specifications for the BayesNetCategorical object

            node\_specs = [

                ('sandy', '', sandy\_prob, [True, False]),

                ('StormProximity', '', storm\_prob, ['Adjacent', 'Diagonal', 'NotNearby']),

                ('air\_quality', 'StormProximity sandy', ppm\_cpt, ['High', 'Medium', 'Low'])

            ]

            # Create the BayesNetCategorical object

            cellbn = BayesNetCategorical(node\_specs)

            # Add the Bayesian network to the dictionary

            bayesnets[(i, j)] = cellbn

    return bayesnets

I understand I did it differently than the project outlined, but as we talked about, I found a way to make it work like this.

1. Report the % of wins and compare it to the one from Programming Assignment 2.

Project Code, 100 runs results:

* Number of victories: 54
* Sandy movement cost average: 88.01

PA2 Code, 100 runs results:

* Number of victories: 47
* Movement cost average: 49.93

Comparing the sample of results from both codes, it’s clear that the Project code, however more costly, is much more successful at completing the task on average. This became obvious through development. The additional understanding of storm probability added an additional layer of knowledge for the agent to utilize and removed some of the luck from the win rate. It did however come at a higher cost, but I believe the higher cost is a result of two things, being forced to repair faults, and the fact that the rover had on average a higher quality of search due to the additional information, meaning that it made more moves on average, because dying instantly based on random chance was much more unlikely.