CS5011 – REPORT(P2)

220026989

1.Introduction

This report wholesomely summarizes the A2Main coursework implementation submitted. It should be noted that all the terminologies, functions and classes are well commented in the Java file and therefore the same is not repeated in the report. All the implementations – P2 to P4 (except P1), are programmed such that it can take any BN (The variables are not restricted to binary).

i. Checklist of implementation.

- ✓ Part 1 – Simple inference

  Simple inference on a given chain is implemented.

- ✓ Part 2 – Variable Elimination without evidence

  Variable elimination to find the required probability with BN's having any number of branching is implemented.

- ✓ Part 3 – General inference on a general BN

  Two advanced Search methods are implemented:

  - An informed Bidirectional search is implemented with a cost function taking snippets of code/ideas from A Star search and is working as expected.
  - An informed triple agent search with heuristics is implemented and is working as expected.

- ✓ Part 4 – Extension

  Validation of the given BN is implemented:

  - Validated the probability values provided.
  - Validated if the BN is DAG or not.

ii. Compilation instructions.

- After navigating into the src folder :
  - *Compilation :*

    ```
    >> javac A2main.java
    ```
  - *Running the code:*

```
>> java A2main <Pn> <Filename.xml>
```

Then the required variables are asked from the user. An example of running the program :

- P1
  Query:
  D:T

- P2
  Query:
  N:T
  Order:
  J,L,K,M,O

- P3
  Query:
  Z:T
  Evidence:
  R:T U:T

- *P4* – only requires the user to enter the run command

2.Design and Implementation

The program mainly uses stride implementation, a concept that prioritizes how fast a random variable changes in a given section of the CPT. Furthermore, I've used the name **category** or **categories** to imply the types/number of variables of a given random variable. All the three parts of the program makes use of a single class. Instead of creating multiple classes for different parts, I found that I could avoid a huge chunk of code repetition, if all are bubbled up in the same class. An important point to be noted although is when the objects – factors (here)a re created it is created with all the combined requirements of the parts. Some of the variables are not used after the initial use.

i. Project structure:

- **A2main.java :** This class is the main class of the program. All the basic implementation is done in this class.
- **Factors.java :** This class has the main properties of the so called factors. This class, for the part1 acts as the probability object while for the rest it shows the characteristics of a factor.

ii. Implementation overview:

- ### General

The A2main class is tasked with reading the xml files. After taking in the file -
sortInputs() function is called. SortInputs() is the basic crucial function that reads the xml file accurately, creates the factor objects and its categories. The categories for a given random variable are the types of variables it can have. Note, that here we treat the factors as probability distributions or CPTs, so they have a mainVariable and a dependent variable(s). Then the basic variables required for the smooth functioning of the program are filled – such as parentVariables(the variables/rvs on which the current CPT is dependent on).
Finally completeAll() and setup() functions are invoked that fill in the strideValues and categoryValues for the specific factor.
Inorder to avoid ambiguity while going through the program it should be made known that:
(Ambiguous variables in Factor class explanation)

- **Categories** - categories is the variable that stores the number of types of variables for each variable in a factor. (i.e – If a factor has n number of rvs in it, then categories is an n-sized arraylist)

- **eachCategories** – While categories keeps track of the categories of value for a variable in a given factor, eachCategories is a public static variable that keeps track of the number of categories of all the random variable in the program.

- **eachCategoryValues** – is a rather simpler variable that keeps the numbering(integer) according to the number of categories. i.e – If a random variable has two categories – True and False, eachCategoryValues keeps the value 1 for True and 2 for False, for the given variable.

- **givenCategories** – store the given values of a variable as specified in the .xml file as it is.

  These variables are important to implement Simple inference/variable elimination with the help of stride for variables that has more than 2 types of categories.

- ### P1 – Simple inference

Here each variable has one parent variable:

I. Initially after the creation of the factorList (or the creation of BN), we find the factor where the mainVariable is the queryVariable (queryFactor). The required value of the variable whose probability is to be found is also converted to the numbering value (as mentioned in the eachCategoryValues above) with the help of givenCategories.

II. The **P1 method** is then called. P1() is a relatively easier method which makes use of the basic probability equation :

    P(B) = P(A) * P(B|A) + (1-P(A)) * P(B|not A)

III. The parent Probability is calculated by recursively calling the same function until a variable which does not have a parent is met.

○ P2 – Variable elimination without evidence

Here the BNs provided can have more than one parent (branching). In this case the user also enters the order of how variable is to be eliminated.

I. Apart from the similar initial steps encountered in the P1 case, we also have a String array that contains the variable elimination order.

II. The **P2()** method is different from P1(). Initially, for each variable in the order array, we find all the factors that has the variable to be eliminated. These factors are temporarily stores and then joined using join() method that returns a combinedFactor. From this combinedFactor the variable to be eliminated is summed out with the help of sumout()method.

III. Finally after going through all the order array variables, the number of factors left in factorList is checked. If there are more than one, these are just joined as they would be of the factors of the same variable. This is then normalized and the required value is found and returned

Briefing of functions used:
(elaborate explanation provided in the program)
- join()
Is a method in the A2main class that takes in an arraylist of factors. The factor which has the maximum number of variables is then found from it (for easiness) and the rest of the factors are multiplied to it with a join function defined in the factor class.

- sumout()
Finally the factor that has to be summed out is given to the method. The method creates a finalFactor and keeps track of the indexes and assignment of the original factor. This is then compared and initialized to the summed out factor except for the summing variable. The values of the summing variable is found and then summed to have the final result.

○ P3 – General inference on general BN

Here we do not have the order of variable elimination but we have evidence provided.

I. Just like the above methods the initial implementation of P3 is the same. We create a separate evidenceVars variable that keeps track of the evidence variables. A method called setNuisanceVars is invoked that finds the nuisance variables for the given case. Now when nuisance variable is not zero, we assign the evidence variable and find the order for order elimination.We use a getOrder method that uses a greedy algorithm to find the factor(or mainvariable) that will have the minimum number of variables associated. This variable is returned and joined with the other factors and summed out. This process is carried out until there are no nuisance variable remaining. Finally a findResult method is invoked which finds the required result according to the evidence provided.

- ### P4 - Extension

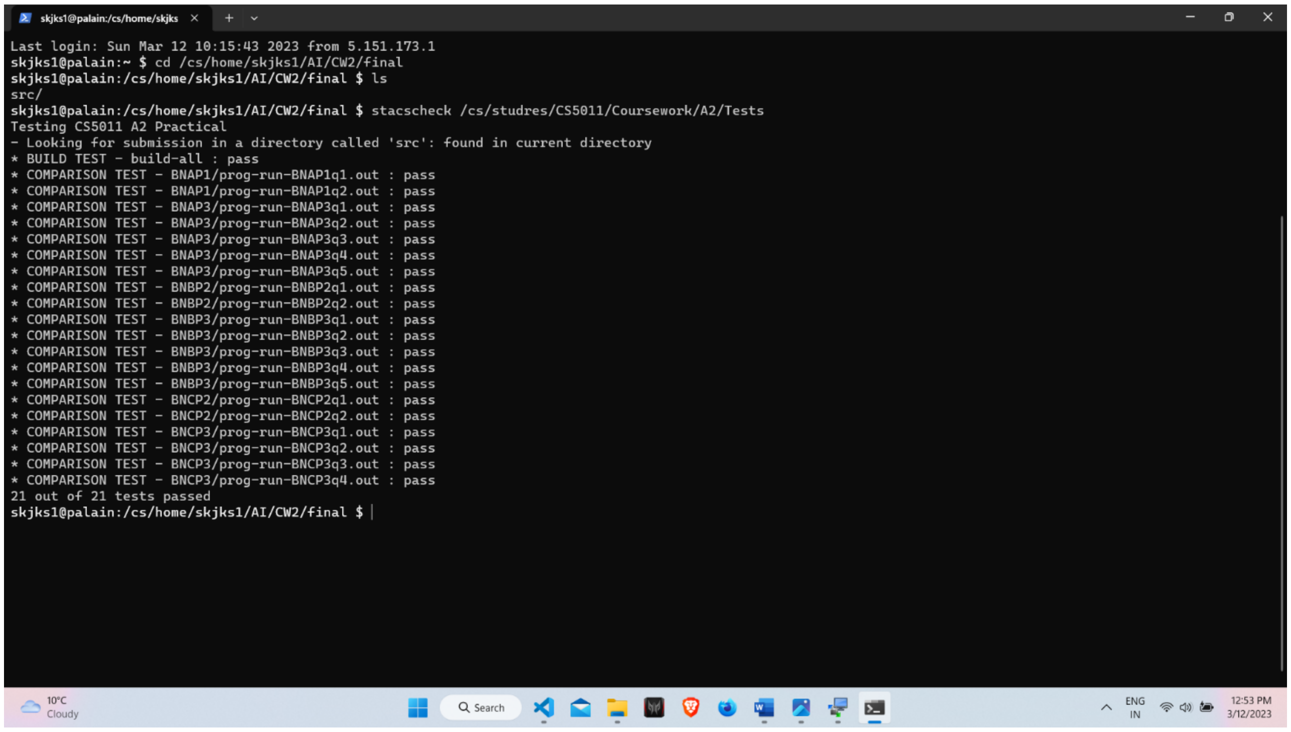
Validation of the BN is taken care.

Initially each input of the probability distribution is checked for any incorrect value(negative or summed out to non-zero). This is done with the basic implementation of categories.

Secondly, we check if the given BN is DAG OR DCG. This is done with the help of the childNodes that exist in the factor object. Each factor object has the list of child factors. We take one factor at a time then, evaluate the children nodes. Here we look at a depth first search algorithm where for a given factor, we go through the child Nodes of the childnodes recursively until we reach the factor that has no childNoDE. If the initial factor is found without reaching this specific node(that has no child node), the given graph is DCG.

3.Test summary

Stacscheck is done with the given src file and all the tests have been passed.



```
Last login: Sun Mar 12 10:15:43 2023 from 5.151.173.1
skjks1@palain:~ $ cd /cs/home/skjks1/AI/CW2/final
skjks1@palain:/cs/home/skjks1/AI/CW2/final $ ls
src/
skjks1@palain:/cs/home/skjks1/AI/CW2/final $ stacscheck /cs/studres/CS5011/Coursework/A2/Tests
Testing CS5011 A2 Practical
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - build-all : pass
* COMPARISON TEST - BNAP1/prog-run-BNAP1q1.out : pass
* COMPARISON TEST - BNAP1/prog-run-BNAP1q2.out : pass
* COMPARISON TEST - BNAP3/prog-run-BNAP3q1.out : pass
* COMPARISON TEST - BNAP3/prog-run-BNAP3q2.out : pass
* COMPARISON TEST - BNAP3/prog-run-BNAP3q3.out : pass
* COMPARISON TEST - BNAP3/prog-run-BNAP3q4.out : pass
* COMPARISON TEST - BNAP3/prog-run-BNAP3q5.out : pass
* COMPARISON TEST - BNBP2/prog-run-BNBP2q1.out : pass
* COMPARISON TEST - BNBP2/prog-run-BNBP2q2.out : pass
* COMPARISON TEST - BNBP3/prog-run-BNBP3q1.out : pass
* COMPARISON TEST - BNBP3/prog-run-BNBP3q2.out : pass
* COMPARISON TEST - BNBP3/prog-run-BNBP3q3.out : pass
* COMPARISON TEST - BNBP3/prog-run-BNBP3q4.out : pass
* COMPARISON TEST - BNBP3/prog-run-BNBP3q5.out : pass
* COMPARISON TEST - BNCP2/prog-run-BNCP2q1.out : pass
* COMPARISON TEST - BNCP2/prog-run-BNCP2q2.out : pass
* COMPARISON TEST - BNCP3/prog-run-BNCP3q1.out : pass
* COMPARISON TEST - BNCP3/prog-run-BNCP3q2.out : pass
* COMPARISON TEST - BNCP3/prog-run-BNCP3q3.out : pass
* COMPARISON TEST - BNCP3/prog-run-BNCP3q4.out : pass
21 out of 21 tests passed
skjks1@palain:/cs/home/skjks1/AI/CW2/final $ |
```

Methods to check the functioning of in-program methods have been provided in commented form in the program.