Lab 2

CS 3793/5233 – Fall 2016 assigned September 13, 2016
Tom Bylander, Instructor due midnight, September 30, 2016

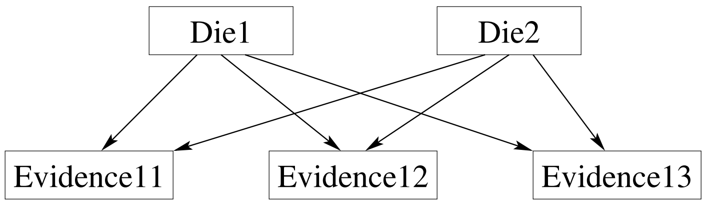
In Lab 2, you will complete a program for playing Dicegame. Dicegame is a game involving two six-sided objects that your instructor has invented and will likely become very popular. The goal is to guess the value of the dice in as few guesses as possible. Before each guess, your program will get three bits of evidence which are probabilistically related to the value of the die. Naturally then, to properly play this game, your program will need to create and use Bayesian networks.

The initial program is ailab2.zip, which you can download from the course web site. This code includes the variable elimination method for making inferences from a Bayesian network (see Section 6.4.1). Your part of the program is to put Bayesian networks together for different game situations. You can play this game by running DiceGame.java. It is expecting you to enter how many games you want to play, so you will need to enter 1 (or some other number) first.

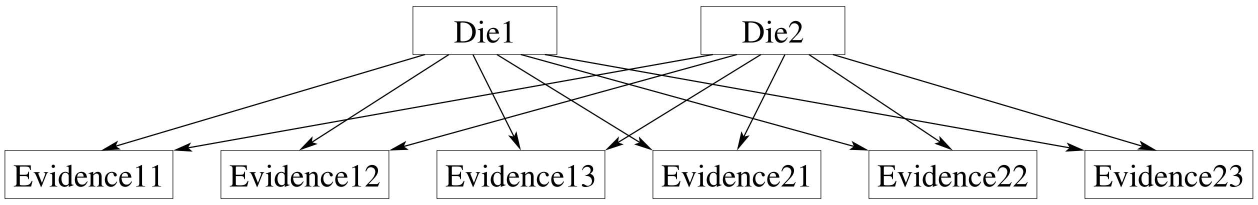
Bayesian Networks for the Game of Die

Dicegame is played in the following way: First, your program receives three bits (see the variables evidence1, evidence2 and evidence3 in DiceGameSolver.java). Then, your program guesses the value of the dice (see the variable guesses in DieGameSolver.java). The game ends if the guess is correct. Else your program receives three more bits, your program makes another guess, and so on.

The way the three bits are generated correspond to the following Bayesian network.



Each round the program gets three additional bits of evidence. This should result in a bigger Bayesian network. For example. the following Bayesian network corresponds to round two.



Each additional round should be modeled with three more evidence variables.

Probability Variables and Tables

Each die has six possible values from one to six, equally likely. All the other variables have two possible values, zero or one. The three evidence bits indicate the following.

1. Is the sum of the dice high? This bit will be one if the sum is 12, zero if the sum is 2, and vary probabilistically inbetween.
2. Is the first die less than, maybe equal to, the second die? This bit will be one if the first die is smaller, zero if the first die is larger, and a coin flip if the two dice are equal.
3. Are both dice odd or maybe just one? This bit will be one if both dice are odd, zero if both dice are even, and a coin flip if one is odd and the other even.

The die corresponds to this probability table. Note that the values in DieGameSolver.java are often off by one because of zero-indexing.

Die	P(Die)
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

The first evidence variable corresponds to this probability table:

		P(Evidence1 | Die1, Die2)	
Die1	Die2	0	1
1	1	1.0	0.0
1	2	0.9	0.1
1	3	0.8	0.2
1	4	0.7	0.3
1	5	0.6	0.4
1	6	0.5	0.5
2	1	0.9	0.1
2	2	0.8	0.2
2	3	0.7	0.3
2	4	0.6	0.4
2	5	0.5	0.5
2	6	0.4	0.6
3	1	0.8	0.2
3	2	0.7	0.3
3	3	0.6	0.4
3	4	0.5	0.5
3	5	0.4	0.6
3	6	0.3	0.7
4	1	0.7	0.3
4	2	0.6	0.4
4	3	0.5	0.5
4	4	0.4	0.6
4	5	0.3	0.7
4	6	0.2	0.8
5	1	0.6	0.4
5	2	0.5	0.5
5	3	0.4	0.6
5	4	0.3	0.7
5	5	0.2	0.8
5	6	0.1	0.9
6	1	0.5	0.5
6	2	0.4	0.6
6	3	0.3	0.7
6	4	0.2	0.8
6	5	0.1	0.9
6	6	0.0	1.0

The second evidence variable corresponds to this probability table:

		P(Evidence2 | Die1, Die2)	
Die1	Die2	0	1
1	1	0.5	0.5
1	2	0.0	1.0
1	3	0.0	1.0
1	4	0.0	1.0
1	5	0.0	1.0
1	6	0.0	1.0
2	1	1.0	0.0
2	2	0.5	0.5
2	3	0.0	1.0
2	4	0.0	1.0
2	5	0.0	1.0
2	6	0.0	1.0
3	1	1.0	0.0
3	2	1.0	0.0
3	3	0.5	0.5
3	4	0.0	1.0
3	5	0.0	1.0
3	6	0.0	1.0
4	1	1.0	0.0
4	2	1.0	0.0
4	3	1.0	0.0
4	4	0.5	0.5
4	5	0.0	1.0
4	6	0.0	1.0
5	1	1.0	0.0
5	2	1.0	0.0
5	3	1.0	0.0
5	4	1.0	0.0
5	5	0.5	0.5
5	6	0.0	1.0
6	1	1.0	0.0
6	2	1.0	0.0
6	3	1.0	0.0
6	4	1.0	0.0
6	5	1.0	0.0
6	6	0.5	0.5

The third evidence variable corresponds to this probability table:

		P(Evidence3 | Die1, Die2)	
Die1	Die2	0	1
1	1	0.0	1.0
1	2	0.5	0.5
1	3	0.0	1.0
1	4	0.5	0.5
1	5	0.0	1.0
1	6	0.5	0.5
2	1	0.5	0.5
2	2	1.0	0.0
2	3	0.5	0.5
2	4	1.0	0.0
2	5	0.5	0.5
2	6	1.0	0.0
3	1	0.0	1.0
3	2	0.5	0.5
3	3	0.0	1.0
3	4	0.5	0.5
3	5	0.0	1.0
3	6	0.5	0.5
4	1	0.5	0.5
4	2	1.0	0.0
4	3	0.5	0.5
4	4	1.0	0.0
4	5	0.5	0.5
4	6	1.0	0.0
5	1	0.0	1.0
5	2	0.5	0.5
5	3	0.0	1.0
5	4	0.5	0.5
5	5	0.0	1.0
5	6	0.5	0.5
6	1	0.5	0.5
6	2	1.0	0.0
6	3	0.5	0.5
6	4	1.0	0.0
6	5	0.5	0.5
6	6	1.0	0.0

The Programming in Lab 2

Although you students would undoubtedly learn more if you had to implement Bayesian networks and variable elimination from scratch, your instructor has decided to try to simplify the creation and operation of Bayesian networks for you. The disadvantage is trying to reverse-engineer the instructor's design while trying to implement the missing pieces. Hopefully, this explanation and the comments in the code will help.

Your Task

Your task is to finish the diceNetwork method in DiceGameSolver.java so it creates a Bayesian network as specified above. More variables and factors need to be added to fully specify the network structure and probabilities. Also, the diceNetwork method needs to tell the Bayesian network what evidence has been observed.

The file BayesianNetworkTest.java shows an example based on a Bayesian network from the textbook. One method in this file shows several examples of running a Bayesian network (that is, calling the eliminateVariables method). Note that the diceNetwork method already contains the calls to the eliminateVariables method.

The Components of ailab2.zip

- `Interact.java` runs `DiceGame.java` (the "environment") at the same time as `DiceGameSolver.java` (the "agent").
- The first thing `DiceGameSolver.java` prints is the number of games to play. `DiceGame.java` then proceeds to officiate multiple games of Dicegame.
- `BayesianNetwork.java` defines a Bayesian network as a set of variables and a set of factors. Some error checking is done. You will need to use the `addVariable` and `addFactor` and `observe` methods. Calling the other methods as needed is already coded. `BayesianNetworkTest.java` shows how examples in the textbook are implemented.
- `Variable.java` defines variables for Bayesian networks. All a variable does is keep track of a name (String) and a set of possible values (an array of Strings). You will need to create `Variable` objects in the `diceNetwork` method. There are two things to keep in mind when using `Variable` objects in this program.
  - For each Bayesian network, you should not create duplicate names for variables.

  - Generally, a value is not referred to by name elsewhere in the program. Instead, an index into the array of values is used to refer to a particular value. For example in `DiceGameSolver.DIE_VALUES`, `"one"` is index 0, `"two"` is index 1, and so on.

- `Factor.java` defines factors as a set of variables and an assignment of a `double` to each combination of values of the variables. Note that each probability table maps to a factor, and that each variable in the Bayesian network corresponds to a table/factor (the probabilities for that variable based on its parents). See `FactorTest.java` and `BayesianNetwork.java` for examples from the textbook.

  Here is an example of assigning a *double* to a combination of values. For example, using $D1$, $D2$ and $E1$ to denote the names of the variables for the dice and the first evidence variable, then one value in the probability table is:

  $$P(E1 = 0 \mid D1 = \text{five}, D2 = \text{one}) = 0.5$$

  and could be formally stated in a factor f as:

  $$f(D1 = \text{five}, D2 = \text{two}, E1 = 0) = 0.5$$

  In `Factor.java`, these assignments are represented as indexes.

  $$f(4, 1, 0) = 0.5$$

  This is because `"five"` and `"two"` are at index 4 and index 1 of `DiceGameSolver.DIE_VALUES`, and the value for the evidence directly correspond to the index.

  The code to correspond to the above assignment in `DiceGameSolver.java` should be:

  ```
  factorForEvidence1.set(0.5, 4, 1, 0);
  ```

  where `factorForEvidence1` is a `Factor` created earlier. Note that this factor requires 72 calls to the set method (a double loop might be useful here). Also note that the probability comes first followed by the indexes. This is because of how Java's variable-length arguments feature works.

Turning in Your Lab

Submit the folder containing all your Java files to Blackboard. Your program should run under the following conditions: after all the .class files are deleted, then something like:

```
javac Interact.java
java Interact
```

should solve the problems.

The score of your lab will primarily depend on the performance of the program. Over 10, 100 and 1000 games, the initial download will average 20.6, 19.05 and 18.471 guesses per game. Your instructor's implementation reduces these averages to 5.2, 4.54 and 4.303.

Testing Your Program

My implementation will produce the following output when it is run on the first 2 games (using 13 as the value of the seed field in DieGame.java).

```
class dicegame.DiceGameSolver 10
class dicegame.DiceGame 0 0 0
class dicegame.DiceGameSolver 4 2
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 0 0 1
class dicegame.DiceGameSolver 2 1
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 0 0 0
class dicegame.DiceGameSolver 4 1
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 1 0 0
class dicegame.DiceGameSolver 3 1
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 1 0 1
class dicegame.DiceGameSolver 6 1
class dicegame.DiceGame right in 5 tries, after 1 games average = 5.000000
class dicegame.DiceGame 0 0 1
class dicegame.DiceGameSolver 5 1
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 1 0 0
class dicegame.DiceGameSolver 6 1
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 1 0 1
class dicegame.DiceGameSolver 6 3
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 0 0 1
class dicegame.DiceGameSolver 6 2
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 0 0 1
class dicegame.DiceGameSolver 4 1
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 1 0 1
class dicegame.DiceGameSolver 4 2
class dicegame.DiceGame wrong try again
class dicegame.DiceGame 0 0 1
class dicegame.DiceGameSolver 4 3
class dicegame.DiceGame right in 7 tries, after 2 games average = 6.000000
```