Bayesian Network assignment

In this assignment you'll create a C++ class that represents a Bayesian network (this is half implemented in `generate.data.cpp`). The main goal of the assignment is to implement an inference algorithm – algorithm that evaluates probability given values of some variables (evidence). Secondary goal is to implement learning.

Step 1: exact inference in Bayesian network. See example below.

Syntax: `ExactInference("1-1--|---10")` – the probability of variable 0 = true AND 2 = true, given 3 is true and 4 is false. That is

$$P(X_0 = 1, X_2 = 1 | X_3 = 1, X_4 = 0)$$

Variable enumeration is user-defined – see `main.cpp`. Notice that you'll need some string processing, you may use my code – see `TrainingData::EstimateProbability(...)`.

Step 2 (brute force)

Implement learning using brute force solution: for all permutations of random variables (that'll be the factorial $n!$ for n variables) construct a network (slides 12–18 in "chapther14a.pdf") and test it.

when deciding whether to add a link, use estimated probabilities.

To test the network you'll have to fix a set of events whose probabilities (for every network) you calculate by 1) sampling (`TrainingData::EstimateProbability`) and by 2) exact inference. The difference between them is the error. Network that has the smallest total error (sum of all differences) in the winner.

BTW – there is an STL function that returns permutations.

Step 2.

A faster approach compared to brute-force. We'll use local search techniques (cs381) to optimize the structure. What we need is

1. neighbor relation – which structures (graphs) are neighbors of the current one

2. evaluation function. How good is the current structure.

Here are some ideas - feel free to modify any

1. neighbor relation. For all search techniques you need to generate children of the current node (**Expand** in search terminilogy). For a network (graph) we may perform the following actions:

   - remove a link (network remains valid)
   - add a link (network may become invalid - check for loops)
   - reverse a link (network may become invalid - check for loops)

   For example, for Buglary domain in `main.cpp` adding a link from Burglary to Earthquake will create a child. Notice, that you have to

   - make sure that new network (i.e. topology) is a valid Bayesian network (that is – no loops)
   - recalculate some of the CPTs (conditional probability tables) – namely for each node with modified number of parents you'll need a new CPT. CPT should be calculated from the training data.

2. evaluation function, a.k.a **fitness** function. Fitness function may use the following quantities:

   - error – how well network predicts the probabilities. Say, randomly choose some number of random strings in the form `"-1---|--1--"`. Evaluate the probability of the event using exact inference and current network topology `BayesianNetwork::ExactInference("-1---|--1--")`, then compare it to the probability of the same event based on corresponding frequencies in the training data (`TrainingData::EstimateProbability("`. Difference of the two values is the error (our goal is to minimize it).
   - topology complexity – what is the total amount of information we need to store the network. That includes: number of links and the sum of all CPT sizes. (our goal is to minimize it)

$$f(network) = \alpha \times \log(complexity) + \beta \times error$$

   notice the log function – this is mostly a hack to ensure that the weights of precision and complexity are on the same scale.

   Remember that local search may get stuck in a local maxima (actually minima in our case), this may be solved using several searches with random restart.

Exact inference example using burglary domain, enumeration B,E,A,J,M and probabilities as in `test0`. This order is the topological order of the corresponding network, so we may proceed without re-ordering. In the case when user-defined

order is not topological (called native in the source code), reordering should be called before step 3.

Exact inference is done in 3 main steps:

1. eliminate conditional probabilities
2. decompose complex events into a union of atomic
3. evaluate probabilities of atomic events using CPTs

Example: **step 1**:

$$P(\text{"}1---|---1-\text{"}) = P(b|j) = \frac{P(bj)}{P(j)} =$$

Example: **step 2**:

$$= \frac{\sum_{\text{atomic events matching } bj} P(event)}{\sum_{\text{atomic events matching } j} P(event)} =$$

$$= \frac{\sum_{x,y,z} P(1xy1z)}{\sum_{u,v,w,t} P(uvw1t)}$$

that is 8 atomic events in the numerator and 16 in the denominator:

$$\sum_{x,y,z} P(1xy1z) = P(\mathbf{1}00\mathbf{1}0) + P(\mathbf{1}00\mathbf{1}1) + P(\mathbf{1}01\mathbf{1}0) + P(\mathbf{1}01\mathbf{1}1) +$$

$$P(\mathbf{1}10\mathbf{1}0) + P(\mathbf{1}10\mathbf{1}1) + P(\mathbf{1}11\mathbf{1}0) + P(\mathbf{1}11\mathbf{1}1)$$

$$\sum_{u,v,w,t} P(1uvwt) = ...$$

Example: **step 3**: Each of the atomic events is calculated using the network structure.

$$P(X_0X_1X_2X_3X_4) = \prod_{i=0}^{4} P(X_i|X_{i-1}X_{i-2}\dots X_0) = \prod_{i=0}^{4} P(X_i|Parents(X_i))$$

example

$$P(10010) = P(b, \neg e, \neg a, j, \neg m) =$$

$$P(b)P(\neg e|b)P(\neg a|\neg e, b)P(j|\neg a, \neg e, b)P(\neg m|j, \neg a, \neg e, b) =$$

remove conditionally independent events from evidence:

$$P(b)P(\neg e)P(\neg a|\neg e, b)P(j|\neg a)P(\neg m|\neg a) =$$

remove negative terms from probabilities:

$$P(b)(1 - P(e))(1 - P(a|\neg e, b))P(j|\neg a)(1 - P(m|\neg a)) =$$

finally – using CPT from `test0`:

$$0.25(1 - 0.1)(1 - 0.8)0.1(1 - 0.2)$$

now do the same calculation 23 more times and we'll get $P(b|j)$.

$$P(\text{"}1 - - - -|- - - 1 - \text{"}) = P(b|j) = \frac{71}{157}$$

If you look back at the previous calculation, you may realize that we had to calculate probabilities for 24 atomic event out 32 total. You may also notice that some of the atomic events in numerator and denominator are the same. There is slightly more efficient way to calculate $P(b|j)$ – find probabilities of 16 atomic events have j in them (that is John did call). Let X be the sum of the 8 that have b, and Y is the sum of the rest 8 (that is that have $\neg b$. Now:

$$P(b|j) = \frac{X}{X + Y}$$

which should be intuitively obvious.