0630395

THE UNIVERSITY *of York*

BSc, BEng, MEng and MMath Examinations, 2006

COMPUTER SCIENCE, COMPUTER SYSTEMS AND SOFTWARE ENGINEERING, COMPUTER SCIENCE AND MATHEMATICS
Part A

ALGORITHMS FOR GRAPHICAL MODELS (AGM)
Open Examination

Issued at:
15 January 2007

Submission due:
7 February 2007

Your attention is drawn to the Guidelines on Mutual Assistance and Collaboration in the Student's Handbook.

All queries on this assessment should be addressed to **James Cussens**.

Your examination number must be written on the front of your submission.
You must not identify yourself in any other way.

Instructions

- All questions should be attempted.
- A submission after the deadline will be marked initially as if it had been handed in on time, but the Board of Examiners will normally apply a lateness penalty.
- Your write-up must be typeset and submitted to the General Office of the Department of Computer Science.
- The programs you write should be submitted electronically through the web page http://www.cs.york.ac.uk/submit. Your programs should be submitted as three files called `agm1.py`, `agm2.py` and `agm3.py`.
- You are permitted to use `gPy` modules in your answers. Only use the version of these modules found in this directory: `/usr/course/agm/gPy/gPy`.
- All extra materials for this assessment can be found at http://www-users.cs.york.ac.uk/~jc/teaching/agm/assessment/06/, henceforth know as the *assessment directory*.
- All solutions must be justified, all programs fully documented. Discuss any possible time versus space compromises. Inefficient solutions will be penalised.
- Unit tests are provided to help you check your programs. Programs which pass the given tests get credit for doing so. To get further marks you need to write programs that are:
  1. efficient in both time and space. (If you decide to effect a trade-off between time and space, explain your reasoning.);

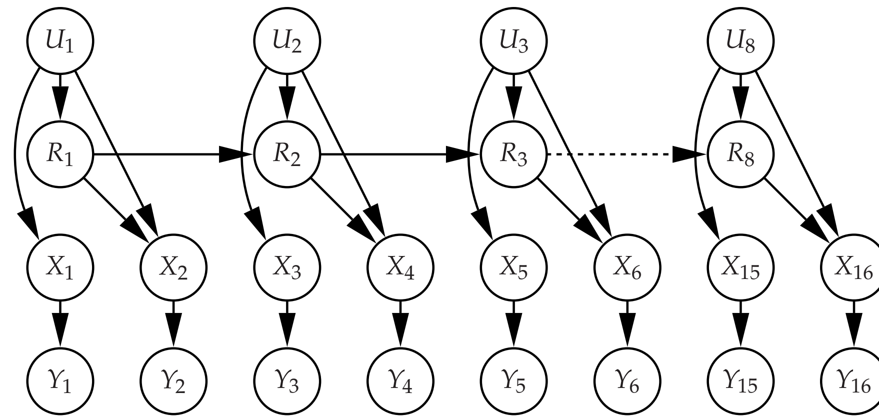

Figure 1: Bayesian network for a simple convolutional code

2. well documented; and
3. which also work on inputs other than those supplied with the tests you have!

1 Convolutional coding using Bayesian networks (30 %)

The answer to this question should be submitted in a file called `agm1.py` Marking scheme: Passing all unit tests = 30 %, Quality of code = 40 %, Quality of write-up = 30 %

This question was inspired by an example from [1]. Error-correcting codes for transmission of digital information over noisy channels can be represented using Bayesian networks. In this question you have to write a Python program which:

1. imports a previously constructed Bayesian network (BN) representing a particular error-correcting code;
2. uses this BN to encode signals; and
3. uses the BN to decode received signals in an attempt to recover the original sent signal.

Throughout, we assume we have a mere 8 bits in our original message. These will be represented by 8 binary variables $U_1, U_2, \dots U_8$. We will actually transmit 16 bits $X_1, X_2, \dots X_{16}$ and the receiver will get 16 bits $Y_1, Y_2, \dots Y_{16}$.

The transmitted bits X_i depend not only on the U_i but also on 8 *register* binary variables $R_1, R_2. \dots R_8$. The dependencies between the variables are those displayed in Fig 1. The conditional probability tables (CPTs) are as follows. $P(U_i = 0) = P(U_i = 1) = 1/2$, for all i, this reflects the fact that 0 and 1 bits are equally likely. Because of noise we have that $P(Y_i = j | X_i = j) = 0.6$ for $i = 1, 2, \dots, 16$ and $j = 0, 1$. In all other cases a child variable is the sum modulo 2 of its parents' values.

In the assessment directory is a Python module called `convcode.py` which defines a `gPy.BNM` object called `codecbn` representing the BN in Fig 1. Your task is to write a Python program called `agm1.py` which imports this object and defines three functions:

encode This takes an 8-bit input (i.e. an instantiation of the U_i) and produces a 16-bit output corresponding to the X_i in Fig 1.

transmit This takes a a 16-bit output corresponding to the X_i in Fig 1 and generates an instantiation of the Y_i. In other words this function emulates transmission of the signal over a noisy channel.

decode This takes a 16-bit input corresponding to an instantiation of the Y_i and produces an 8-bit output corresponding, for each U_i, to the most likely instantiation of that U_i.

The bit sequences should be represented as sequences of 0 and 1 integers. Your functions should be usable no matter how many times they are called, so be careful if your functions change any mutable objects.

How to run the unit tests for this question:

1. Copy `agm1_test.py` from the assessment directory to the directory containing your answer called `agm1.py`.
2. Type `python agm1_test.py`

2 Intelligent variable elimination (35 %)

The answer to this question should be submitted in a file called `agm2.py` Marking scheme: Passing all unit tests = 20 %, Quality of code = 40 %, Quality of write-up = 40 %

The variable elimination algorithm you have seen previously during the module, i.e. the method `HM.variable_elimination`, is actually a very naive version of variable elimination. Specifically, it does not take advantage of conditional independence effectively. In this question you are required to write an improved variable elimination algorithm.

For example, consider a joint distribution over some variables where all variables are independent and so the distribution is represented by a product of univariate factors. Clearly, no work is required to get the marginal of any variable, but the variable elimination algorithm in `gPy` will pointlessly sum out the other variables.

Similarly suppose we have a distribution where the variables in set **S** are instantiated and we want to use variable elimination to get the marginal distribution over a single variable X. If $X \perp \mathbf{Y}|\mathbf{S}$ for a set of variables **S** then your improved variable elimination should take advantage of this to be more efficient than the naive version.

Specifically, define a new class called `HM_q2` which is a subclass of `HM` (a class defined in `gPy.Models`) and which has a single method defined for it called `clever_variable_elimination`. The end result of calling this method should be identical to using the existing `variable_elimination` method in `gPy.Models`. The difference should be that it is more efficient.

How to run the unit test for this question:

1. Copy `agm2_test.py` from the assessment directory to the directory containing your answer called `agm2.py`.
2. Type `python agm2_test.py`

Your write up should provide an account of when this more sophisticated approach makes a difference, and also by how much.

3 Gibbs sampling (35 %)

The answer to this question should be submitted in a file called `agm3.py` Marking scheme: Passing all unit tests = 20 %, Quality of code = 60 %, Quality of write-up = 20 %

The basic task here is implement the Gibbs sampling algorithm. Specifically, define a new class called `HM_q3` which is a subclass of `HM` and which has a single method defined for it called `gibbs_sample`. The first line of the method definition should be:

```
def gibbs_sample(self,iterations,burnin):
```

where:

1. `self` is the `HM_q3` object;
2. `iterations` is the number of iterations of the Gibbs sampling algorithm you want to return; and
3. `burnin` is the number of initial iterations you want to throw away.

The method should return a tuple of tuples representing the instantiations. You should provide evidence that your Gibbs sampler approximates probabilities better the longer the run. Give at least one example where Gibbs sampling does not work at all.

How to run the unit test for this question:

1. Copy `agm3_test.py` from the assessment directory to the directory containing your answer called `agm3.py`.
2. Type `python agm3_test.py`

References

[1] Steffen L. Lauritzen. Some modern applications of graphical models. In Peter J. Green, Nils Lid Hjort, and Sylvia Richardson, editors, *Highly Structured Stochastic Systems*, pages 13–32. OUP, Oxford, 2003.