(http://www.ox.ac.uk/)

FRANK WOOD

)X.AC.UK/~FWOOD/TEACHING/AIMS_CDT_ML/HOMEWORK/INDEX.HTML)

C.UK/~FWOOD/TEACHING/AIMS_CDT_ML/PEOPLE/INDEX.HTML)

JK/~FWOOD/TEACHING/AIMS_CDT_ML/INDEX.HTML)

ASSIGNMENT 1: BELIEF PROPAGATION

In this assignment you will implement sum-product belief propagation on discrete graphical models.

Background reading: Bishop chapter 8, MacKay chapters 16 and 26

GETTING STARTED

- Download the skeleton code for the assignment (hw_1.tar.gz)
- Extract the downloaded material in an appropriate folder, something like `~/Documents/AIMS_CDT_ML/HW1/`
- Open MATLAB and navigate to the folder containing the downloaded material

Graph data structures are difficult to deal with unless you use programing techniques including recursion and object orientation.

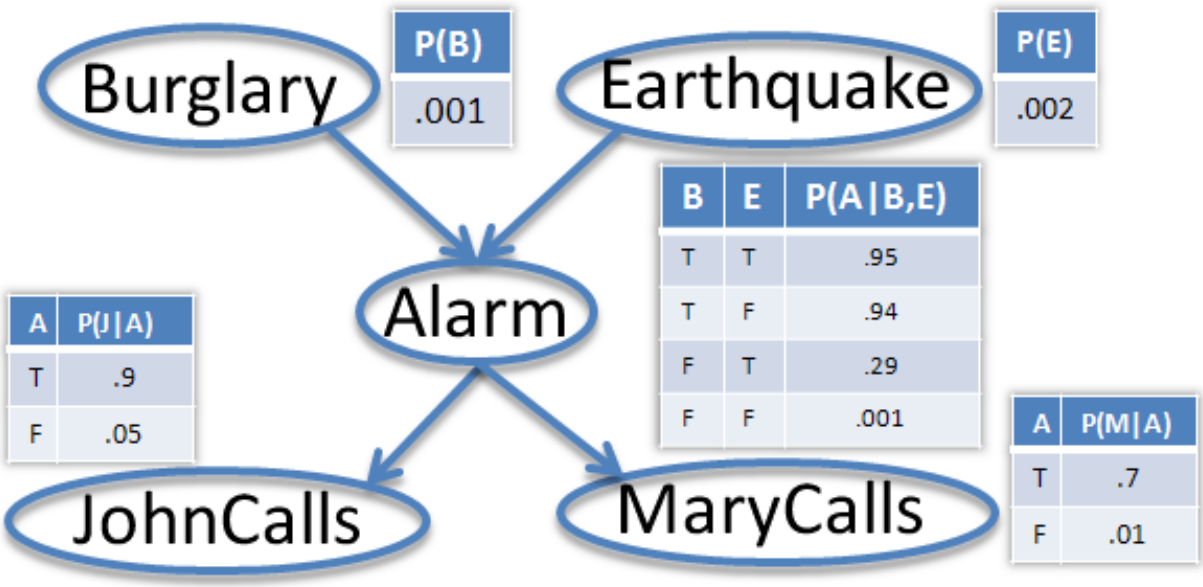
Recursion in computer science is the idea of writing functions that call themselves. When writing general functions to operate on graphs the concept is useful if you want to do an operation, such as message passing, for every node in a graph.

Object oriented programing is a paradigm where you write code to define generic objects and then "instantiate" instances of those objects to perform a task. Generic object definitions are called classes. For instance, you might write code to define a variable node and then create a graphical model by instantiating the appropriate number of them. Objects usually have fields and methods associated with them. Fields are properties, such as if a node has been observed or not or the messages a node has received. Methods are functions associated with the object which can act on the properties of the object and can take outside arguments. For example, it would be useful to have a method which can act on the internal properties of the object and return the outgoing message to a particular factor node. The node will have as properties the most recent messages it has received and can then calculate the correct message to return.

PART 1: SMALL NETWORK, EXPLICIT IMPLEMENTATION

The first part of the assignment consists of implementing key parts of the sum-product algorithm for inference in a discrete graphical model. Only one type of object is used in this code base, namely the *node* object. The node class file does not need to be edited, but you should look at it to understand its properties and methods.

Every variable is binary and so when passing messages the first element of the column vector should be proportional to the probability of `TRUE` and the second number proportional to the probability of `FALSE`. The network you are implementing and the appropriate probability tables are



[figure credit: http://learning.cis.upenn.edu/cis520_fall2009/index.php?n=Lectures.BayesNets (http://learning.cis.upenn.edu/cis520_fall2009/index.php?n=Lectures.BayesNets)]

This graphical model structure is set up in `main_alarm_network.m`. At the bottom of that file you can see an example of inference in the model with no values set. Your tasks are

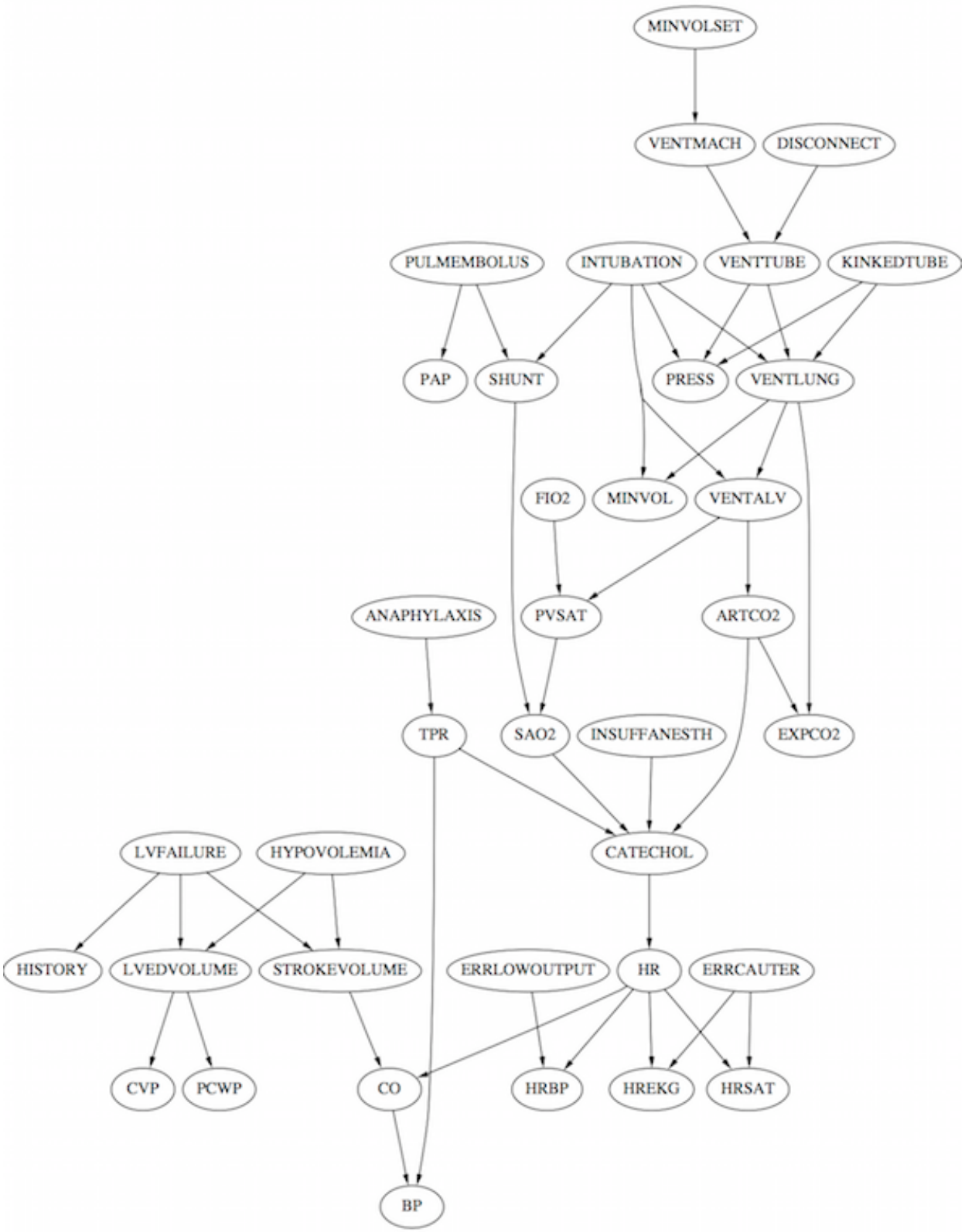
1. Edit this file to calculate the probability there was a burglary given that John called.
2. Calculate the exact conditional distributions by explicitly calculating the probability of every configuration of the variables and then summing out the appropriate variables.

The files you are responsible for filling out:

- `explicit_algorithm/main_alarm_network.m`
- `explicit_algorithm/get_marginal_distribution.m`
- `explicit_algorithm/get_message_fn_aj.m`
- `explicit_algorithm/get_message_fn_am.m`
- `explicit_algorithm/get_message_fn_b.m`
- `explicit_algorithm/get_message_fn_bea.m`
- `explicit_algorithm/get_message_fn_e.m`
- `explicit_algorithm/get_message_vn.m`

PART 2: LARGE NETWORK, OBJECT-ORIENTED IMPLEMENTATION

The second part of the assignment consists of implementing the key parts of the message passing algorithm in a more scalable object oriented setting. The graph is set up for the large ALARM; a network for monitoring patients in intensive care (see http://compbio.cs.huji.ac.il/Repository/Datasets/alarm/alarm.htm (http://compbio.cs.huji.ac.il/Repository/Datasets/alarm/alarm.htm)).



The graph is set up for you in the .m-file `object_oriented_algorithm/main_big_alarm_net.m`. This code base makes extensive use of objects. An object called node is defined in the folder `@node` and two specific nodes are then refined, namely `@factor_node` and `@variable_node`. The refined objects inherit properties and methods from the general `@node` object unless they are specifically overwritten. You will need to familiarize yourself will all three objects to be able to do the project. There is also one other object called `@factor`; a factor object is an object to hold the factor matrix, essentially the probability tables. Finally, you are provided a utility function called `multilinear_product.m` which you may find useful.

You should read the comments at the top of the function which describes its functionality. The last thing you need to know is that if f is the factor matrix inside the factor object `fn_a_b_c` then $f(1,3,2)$ is the factor value of $f(a=1,b=3,c=2)$. Inside each factor node object, the list of neighboring objects is listed in exactly the same way for all factor nodes. That is, for `fn_a_b_c` the neighboring nodes list will be { vn_a, vn_b, vn_c }.

1. The first task in this section of the homework will be to understand the code.
2. Then, you will need to implement the method `getMessage(obj, to_unid)` in both the `factor_node` and `variable_node` objects. This will take some creativity and may require you to use a recursive function or make use of the eval function in MATLAB. You also need to implement the method `getMarginalDistribution(obj)` in the `variable_node` class definition.
3. Finally, set up the main file to calculate the probability of *KinkedTube* given *SaO2* is set to 1, *BP* is set to 1, *ArtCO2* is set to 1, *Press* is set to 2, and *ExpCO2* is set to 1.

Because the network is actually loopy, we have to perform loopy inference here, but it is not critical that you understand how this is implemented. At the bottom of the file `object_oriented_algorithm/main_big_alarm_net.m` you can see an example of inference in the model after setting some specific values.