Probabilistic Graphical Models and Inference
CS181 Assignment 4

Lucas Freitas and Angela Li

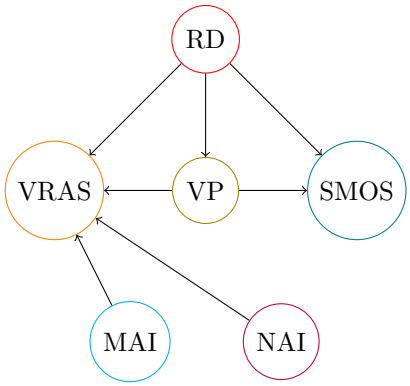
April 13, 2013

Problem 1

(a) From the described attributes, we can suppose that:

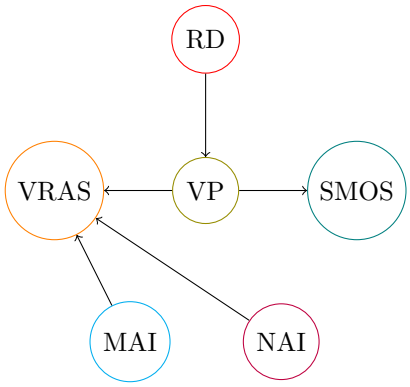
- **RD**: recent downloads from untrusted websites could have virus on them, influencing on the attributes **SMOS**, **VRAS**, and **VP**.
- **SMOS** would not influence any attributes. It is just a result of other attribute being true.
- **VRAS** also does not influence any other attribute.
- **MAI** has influence on **VRAS** being true, since a virus is only reported if an antivirus is present in the computer.
- **NAI** same as **MAI**.
- **VP** influences on **SMOS** and **VRAS**, since those attributes are (usually) only true when a virus is present in the computer. For this item, we consider that silly messages come from virus makers, not as alerts from the anti virus.

From the relations above:



Problem 1 (continued)

If we analyze the graph, we notice that we have some cases of $A \to B \to C$, followed by $A \to C$, such as **RD** $\to$ **VP** $\to$ **VRAS** and **RD** $\to$ **VRAS**. We can delete the $A \to C$ edges if we consider that C is dependent on A from the former relation.



That gives us the following probability tables (notice that **RD, MAI**, and **NAI** do not have probability tables because they do not have nodes pointing towards them):

VRAS probability table:

Let's consider m and n to be the noise for **MAI** and **NAI**.

VP	**MAI**	**NAI**	$P(\textbf{VRAS}) = 1$
0	0	0	0
0	0	1	m
0	1	0	n
0	1	1	$m + n$
1	0	0	0
1	0	1	$1 - m$
1	1	0	$1 - n$
1	1	1	$1 - (n + m)$

VP probability table:

Let's consider t to be the probability of no virus even when recent downloading from untrusted website (noise of **RD**), and u to be the probability of a virus when not having recent downloads from untrusted websites. Notice that if we consider that **RD** is the only factor influencing **VP** and that there are no external noises, then u is 0.

RD	$P(\textbf{VP}) = 1$
0	u
1	$1 - t + u$

Problem 1 (continued)

SMOS probability table:

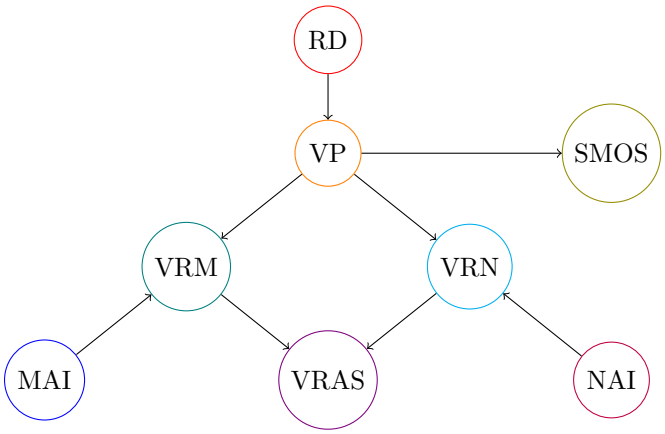
Let's consider p to be the probability of no silly messages on the screen, even when there is a virus, and that there will not be any silly messages if there are no viruses in the computer (if we consider that there might be, we can just another parameter like in the previous table).

VP	$P(\textbf{SMOS}) = 1$
0	0
1	1-p

(b) Two edges that we might consider adding would be from **MAI** and **NAI** to **SMOS**. In our previous item, we consider that silly messages come from the virus maker, not from the anti virus. If we consider that the anti virus might also create those alerts for users when a virus is detected, we might want to add those edges. Besides, we might not want to delete the $A \to C$ relations described in the previous item.

We might imagine, for instance, that maybe the user recently downloaded a program from an untrusted website that is not a virus, but pranks the user displaying silly messages, or making antiviruses think that the computer is infected. In that case, **VRAS** would not be dependent on **RD**, and we should add the edge back.

(c) Adding the two new attributes, we just need to change two things: **VP** now influences both **VRM** and **VRN**, which both influence **VRAS**. That means that the edge from **VP** to **VRAS** should be deleted. Secondly, the edges from **MAI** and **NAI** to **VRAS** should also be moved to **VRM** and **VRN** respectively. That gives us the graph:



And the probability tables:

VRM probability table:

Let's consider m to be the noise for **MAI**.

VP	**MAI**	$P(\textbf{VRM}) = 1$
0	0	0
0	1	m
1	0	0
1	1	$1 - m$

Problem 1 (continued)

VRN probability table:

Let's consider n to be the noise for **NAI**.

VP	**NAI**	$P(\mathbf{VRN}) = 1$
0	0	0
0	1	n
1	0	0
1	1	$1 - n$

VRAS probability table:

VRN	**VRM**	$P(\mathbf{VRAS}) = 1$
0	0	0
0	1	1
1	0	1
1	1	1

(d) The inclusion of nodes **VRM** and **VRN** is definitely helpful for the modeling process. Notice how large and complex the table for **VRAS** in (a) is compared to the one in (c). The 8×3 table from (a) is simplified into two much simmler 4×2 tables for **VRN** and **VRM** and an OR table for **VRAS**. Although that generates a deeper graph, the fact that nodes have less incoming edges makes the model easier to deal with, and states are more explicit and helpful for the model.

Problem 2

a) A variable X is condtionally independent of A given B if X is d-separated from A by B; that is, if all paths from X to A in the directed graphical model are blocked.

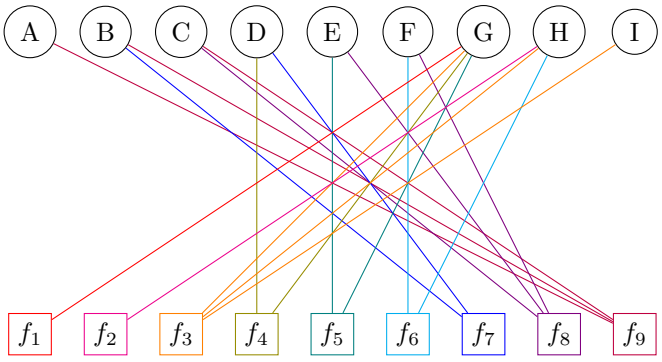
- There are no variables in directed graphical model (a) that are independent of A, given knowledge of J, since there exists no variable X such that all possible paths between X and A are blocked.
- C and F are both independent of A, given knowledge of J, since all possible paths from C or F to A are blocked by I (as it is a head-to-head node and is unobserved). For every other variable X, there is at least one unblocked path from X to A, so X is not conditionally independent of A, given knowledge of J.

b) The joint distribution of a directed graph of n nodes is a product of n conditional distributions, where the ith distribution is only conditioned on the parents of node i.

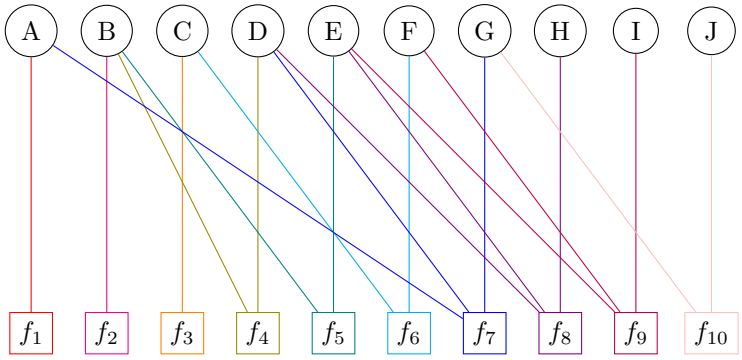
- $p(A, \cdots, I) = f_1(G)\ f_2(H)\ f_3(G, H, I)\ f_4(D, G)\ f_5(E, G)\ f_6(F, H)\ f_7(B, D)\ f_8(C, E, F)\ f_9(A, B, C)$.
- $p(A, \cdots, J) = f_1(A)\ f_2(B)\ f_3(C)\ f_4(B, D)\ f_5(B, E)\ f_6(C, F)\ f_7(A, D, G)\ f_8(D, E, H)\ f_9(E, F, I)\ f_{10}(G, J)$.

c)

- A factor graph representation of (a) is shown below:



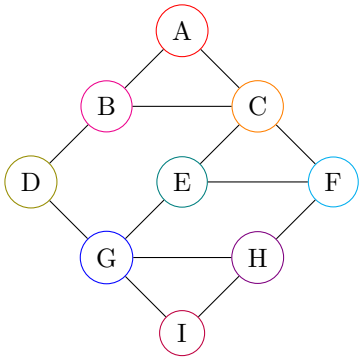
- A factor graph representation of (b) is shown below:



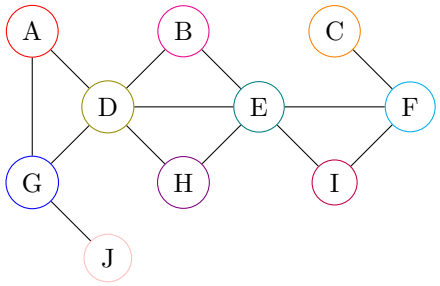
Problem 2 (continued)

(d) We can draw undirected graphical model representations of (a) and (b) by taking the original set of variables as singleton nodes, and drawing an (undirected) edge between them if they are connected by a factor in the factor graph.

- An undirected graph for (a) is shown below:

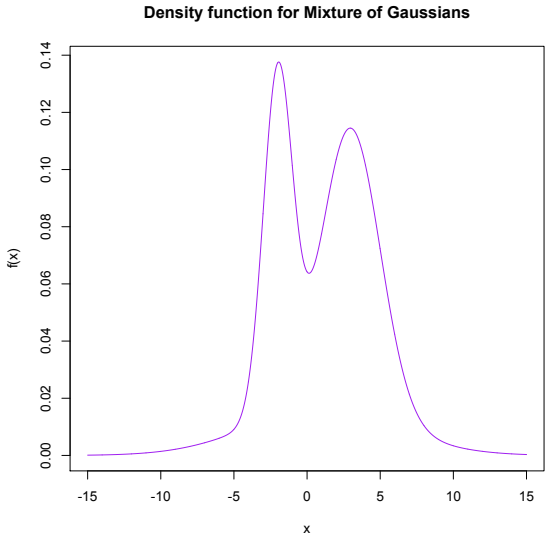


- An undirected graph for (b) is shown below:



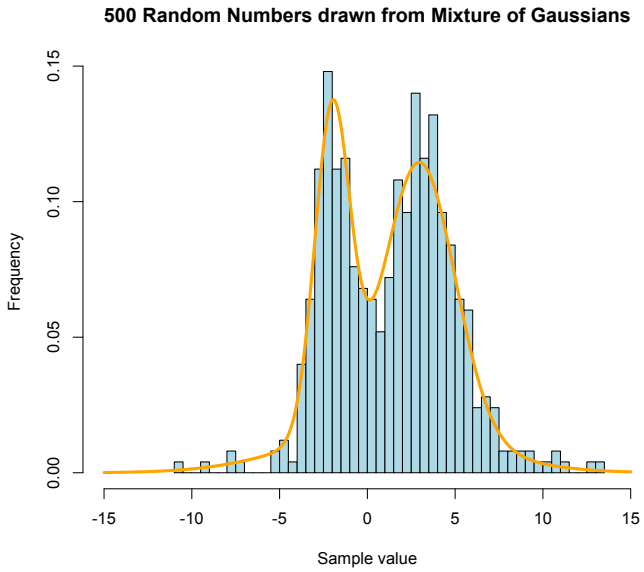
Problem 3

(a) Graph for the probability density function $f(x) = 0.2 \cdot \mathcal{N}(x|\mu = 1, \sigma^2 = 25) + 0.3 \cdot \mathcal{N}(x|\mu = -2, \sigma^2 = 1) + 0.5 \cdot \mathcal{N}(x|\mu = 3, \sigma^2 = 4)$:



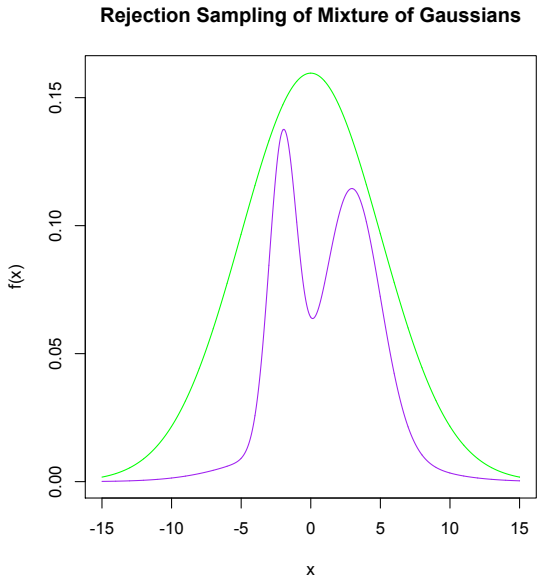
(b) We know how to generate data from Gaussian distributions, and as the provided distribution is a mixture of Gaussians, we can simply sample from the individual Gaussians according to their weights. That is, for every data point, we will sample with probability 0.2 from the distribution $\mathcal{N}(x|\mu = 1, \sigma^2 = 25)$, with probability 0.3 from the distribution $\mathcal{N}(x|\mu = -2, \sigma^2 = 1)$, and with probability 0.5 from the distribution $\mathcal{N}(x|\mu = 3, \sigma^2 = 4)$.

Implementation code can be found in the appendix (at the end of this document), and a histogram of the sampled data is provided below:

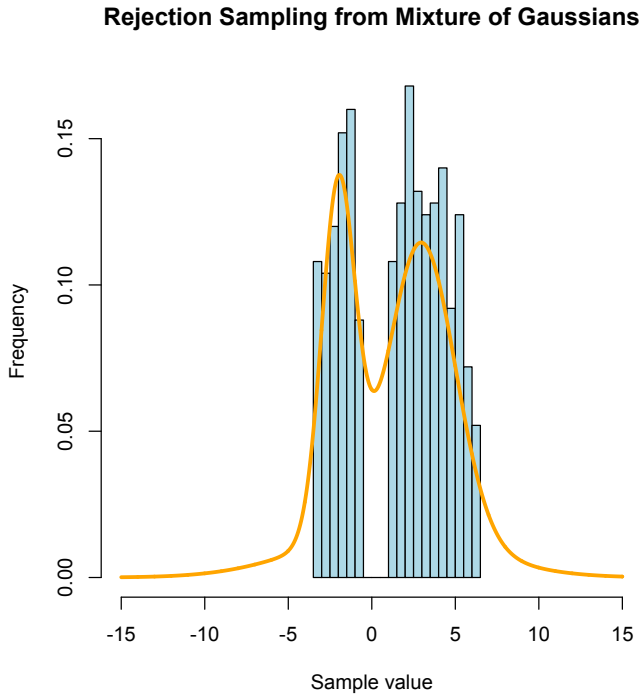


Problem 3 (continued)

(c) To find the upper bounding function, we initially set μ to be aroung 1 and then ajusted c and σ to fit the misture of Gaussians below the curve, and not let the upper bound be too loose. After several adjustments, we decided on using $c \cdot q(x) = 2 \cdot \mathcal{N}(x|\mu = 0, \sigma^2 = 25)$ as our upper bounding function. The mixture of Gaussians indeed fits below the chosen function:



Implementation code can be found in the appendix (at the end of this document), and a histogram of the sampled data is provided below:

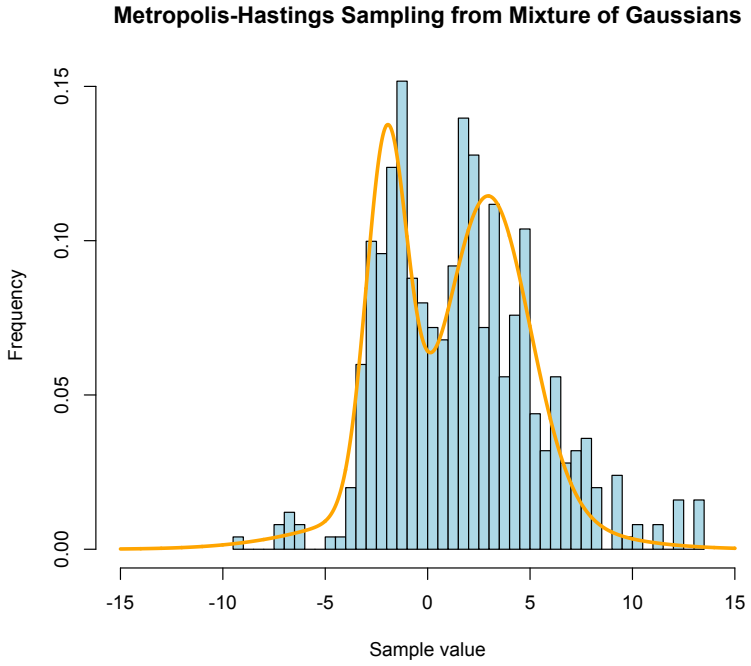


Problem 3 (continued)

The algorithm is good at identifying the two big humps of the mixture of Gaussians, but is not not good at representing values with low frequency, specially values below -4 or above 6, and the values around 0, where we transition from one hump to the other. That shows that rejection sample is fairly good at finding the average shape of a distribution, but does not provide a very accurate aproximation of the distribution for not very frequent values.

(d) After implementing the Metropolis-Hastings algorithm (using a simple Gaussian centered at 1 as the original proposal), we tested several variances to determine which provided a set of points that most resembled the Gaussian PDF, and settled on $\sigma^2 = 7$, which gave an average acceptance rate of $\approx 26\%$. Trying lower variances gave a higher acceptance rate, but a distribution of points that seemed to miss the separation of the two humps, and trying higher variances seemed to result in too many rejections to be useful.

Implementation code can be found in the appendix (at the end of this document), and a histogram of the sampled data is provided below:



Appendix: Python Code

Direct Sampling

```
import random

def mixture(filename, k):
    file = open(filename, "w")
    for i in range(k):
        n = random.randrange(0,10)
        if n < 2:
            file.write(str(random.gauss(1,5)) + ",\n")
        elif n < 5:
            file.write(str(random.gauss(-2,1)) + ",\n")
        else:
            file.write(str(random.gauss(3,2)) + ",\n")
    file.close()

mixture("points.txt", 500)
```

Rejection Sampling

```
import math
import random

# normal PDF
def N(x, mean, var):
    num = math.exp(-(float(x) - float(mean)) ** 2 / (2 * var))
    denom = (2 * math.pi * var) ** .5
    return num / denom

# mixture of gaussians
def mixture(x):
    return 0.2 * N(x, 1, 25) + 0.3 * N(x, -2, 1) + 0.5 * N(x, 3, 4)

# rejection sampling
def sample(filename, k, c, mu, sigma):
    file = open(filename, "w")
    i = 0
    random.seed()
    while i < k:
        x = c * random.gauss(mu, sigma)
        if N(x, mu, sigma * sigma) <= mixture(x):
            file.write(str(x) + ",\n")
            i += 1
    file.close()

sample("data-rejection.txt", 500, 2, 0, 5
```

Metropolis-Hastings Sampling

```
import math, random, sys

file = open("points.txt", "w+")

# target distribution
def pi(x):
    return 0.2 * N(x, 1, 25) + 0.3 * N(x, -2, 1) + 0.5 * N(x, 3, 4)

# normal PDF
def N(x, mean, var):
    num = math.exp(-(float(x) - float(mean)) ** 2 / (2 * var))
    denom = (2 * math.pi * var) ** .5
    return num / denom

# acceptance probability
def acceptance_pr(x, x_prime):
    return min(1, pi(x_prime) / pi(x))

# run metropolis-hastings
def m_h(var):
    accepted = 0
    x = 1
    file.write(str(x) + ",\n")

    # run 500 iterations, plus burn-in phase of 500 iterations
    for i in range(500):
        x_prime = random.gauss(int(x), int(var))
        alpha = acceptance_pr(x, x_prime)
        print alpha
        u = random.uniform(0, 1)
        if u < alpha:
            x = x_prime
            accepted += 1
        file.write(str(x) + ",\n")

    # show our acceptance rate
    print "Acceptance rate: {0}".format(float(accepted) / 1000)

def main():
    var = int(sys.argv[1])
    m_h(var)

if __name__ == "__main__":
    main()
```