**CSE 555**

**Assignment 2**

**Name:Rajiv Ranjan**

**Id:50249099**

Problem 1: Draw the moral graph, triangulated graph and the junction tree. Explain why the "running intersection property" is satisfied in your junction tree.

Solution 1:

Part1:

First of all, we try to do a network setup. One simple way of specifying a model for the chest clinic example is as follows. First we specify conditional probability tables with values as given in Lauritzen and Spiegelhalter (1988). This can be done with array() or as here with the cptable() function, which offers some additional features.

library(gRain)

require(gRbase)

require(Rgraphviz)

yn <- c("yes","no")

a <- cptable(~asia, values=c(1,99), levels=yn)

t.a <- cptable(~tub+asia, values=c(5,95,1,99), levels=yn)

s <- cptable(~smoke, values=c(5,5), levels=yn)

l.s <- cptable(~lung+smoke, values=c(1,9,1,99), levels=yn)

b.s <- cptable(~bronc+smoke, values=c(6,4,3,7), levels=yn)

e.lt<cptable(~either+lung+tub,values=c(1,0,1,0,1,0,0,1),levels=yn)

x.e <- cptable(~xray+either, values=c(98,2,5,95), levels=yn)

d.be<cptable(~dysp+bronc+either,values=c(9,1,7,3,8,2,1,9),levels=yn)

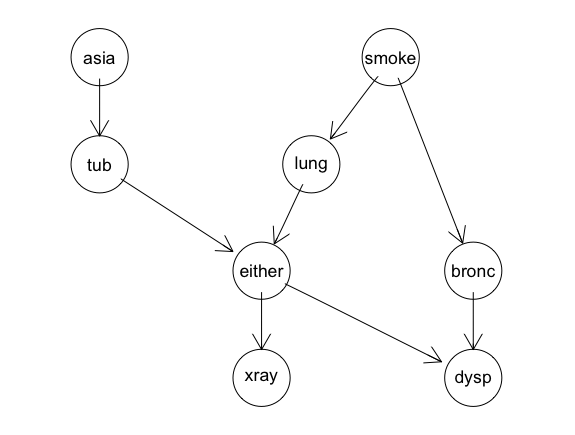
A network is created with the function grain() which returns an object of class grain:

plist <- compileCPT(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))

grn1 <- grain(plist)

plot(grn1)

Output is as follows:



The compileCPT() function does some checking of the specified CPT’s. (For example, it is checked that the graph defined by the CPT’s is acyclic. Furthermore, the specification of t.a gives a table with four entries and the variable tub is specified to be binary. Hence it is checked that the variable asia is also binary.) The object plist is a list of arrays and it is from this list that the grain object is created.

2nd Step : Compilation—Finding the Clique Potentials

A grain object must be compiled and propagated before queries can be made. These steps are performed by the querygrain() function if necessary, but for some purposes it is advantageous to perform them explicitly. Compilation of a network is done with the compile() method for grain objects:

grn1c <- compile(grn1)

Compilation of a grain object based on CPTs involves the following steps: First it is checked whether the list of CPTs defines a directed acyclic graph (a DAG). If so, then the DAG is created; it is moralized and triangulated to form a chordal (triangulated) graph. The CPTs are transformed into clique potentials defined on the cliques of this chordal graph. The chordal graph together with the corresponding clique potentials are the most essential components of a grain object, and one may indeed construct a grain object directly from a specification of these two components.

We again consider the Bayesian network. The factorization

into a clique potential representation follows by simply noticing that in each of the conditional probability tables can be considered a function of the variables it involves. These potentials are simply non-negative functions. The dependence graph of the Bayesian network is derived from the potentials. For example, the presence of the term p(xD |xE,xB) implies that there must be edges between all pairs in {D,E,B}. Algorithmically, the dependence graph can be formed from the DAG by moralization: The moral graph of a DAG is obtained by first joining all parents of each node by a line and then dropping the directions on the arrows. For the chest clinic example, the edges between tub and lung, and between either and bronc are added.

The next step is to triangulate the dependence graph if it is not already so by adding additional edges, so-called fill-ins. This is done to enable simple computation of marginals from the clique potentials, cf. Finding an optimal triangulation (in terms of a minimal number of fill-ins) of a given graph is NP-complete, but various good heuristics exist. The gRbase package implements a Minimum Clique Weight Heuristic method inspired by Kjærulff (1990). Two possible fill-ins are the edge between lung and bronc, and the edge between

either and smoke. The triangulated graph is also a dependence graph for, the graph just conceals some conditional independence restrictions implied by the model. The steps described above can alternatively be carried out separately.

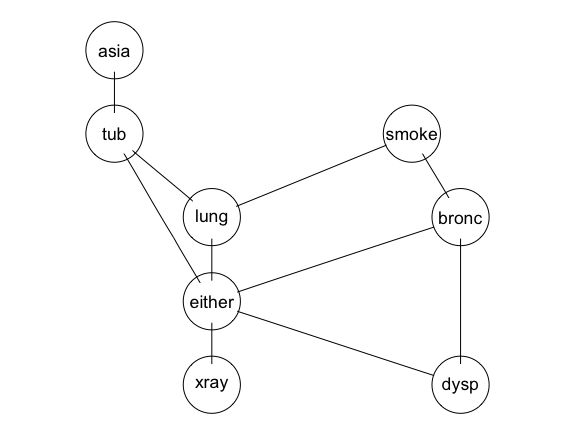
g <- grn1$dag

plot(g)

mg <- moralize(g)

plot(mg)

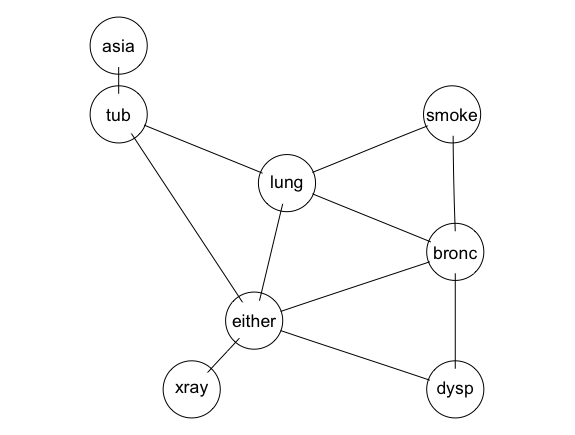
The output of the moralized graph is as follows:



tmg <- triangulate(mg)

plot(tmg)

The output of the finally triangulated graph is as follows:



Triangulated graph

An ordering C1,...,CT of the cliques of a graph is a RIP ordering if Sj = (C1 ∪ ··· ∪ Cj−1) ∩ Cj is contained in one (but possibly several) of the cliques C1,...,Cj−1, obtained with:

rip(tmg)

The output is as follows:

cliques

1 : tub asia

2 : either tub lung

3 : bronc lung either

4 : smoke lung bronc

5 : dysp bronc either

6 : xray either

separators

1 :

2 : tub

3 : lung either

4 : lung bronc

5 : bronc either

6 : either

parents

1 : 0

2 : 1

3 : 2

4 : 3

5 : 3

6 : 5

Picking a particular clique, say Ck, with Sj ⊆ Ck and naming this at the parentclique of Cj , with Cj being the child of Ck, organizes the cliques of the triangulated graph in a rooted tree with the cliques as nodes and arrows from parent to child. We call Sj the separator and Rj = Cj \Sj the residual, where S1 = ∅. The junction tree is formed by ignoring the root and the directions on the edges. It is a tree with the

property that for any pair (A,B) of cliques and any clique C on the unique path between A and B it holds that A ∩ B ⊆ C. It can be shown that the cliques of a graph can be organized in a junction tree if and only if the graph is triangulated.

The junction tree can be displayed by plot(), where the numbers on the nodes refer to the clique numbers in the RIP-ordering. Other RIP-orderings of the cliques can be found by choosing an arbitrary clique

as the first and then numbering the cliques in any way which is increasing as one moves outward from the first clique in this tree. For example C3,C2,C5,C1,C6,C4 would be another RIP-ordering.

The functions p(iv |ipa(v)) are hence defined on complete sets of the triangulated graph. For each clique C we collect the conditional probability tables p(iv |ipa(v)) into a single term ψC(iC) by multiplying them. Triangulation may have created cliques to which no CPT corresponds. For each such clique the corresponding potential is identically equal to 1. Thus we have obtained the clique potential representation of p(iV ) as p(iV ) = product of( j=1(ψCj(iCj ))). The representation is the fundamental representation for the subsequent

computations. As such, a DAG and a corresponding factorization as in is just one way of getting to the representation in and one may alternatively specify this directly as shall be illustrated.

grn1c <- compile(grn1)

summary(grn1c)

plot(grn1c,type="jt")

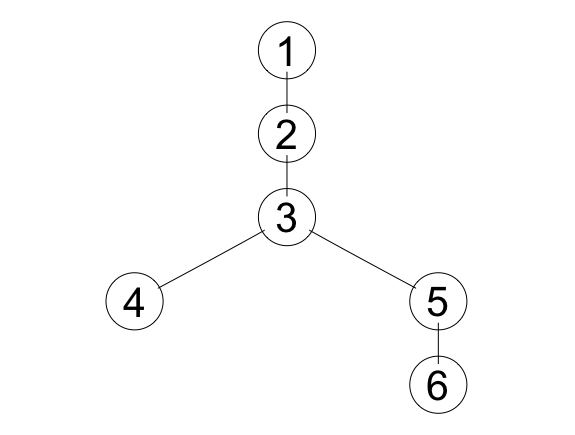


Fig: the junction tree as created by the moralized triangulated graph

Showing the different cliques in the graph

The cliques are as follows:

str( jTree( bnet$ug )$cliques )

List of 6

$ : chr [1:2] "asia" "tub"

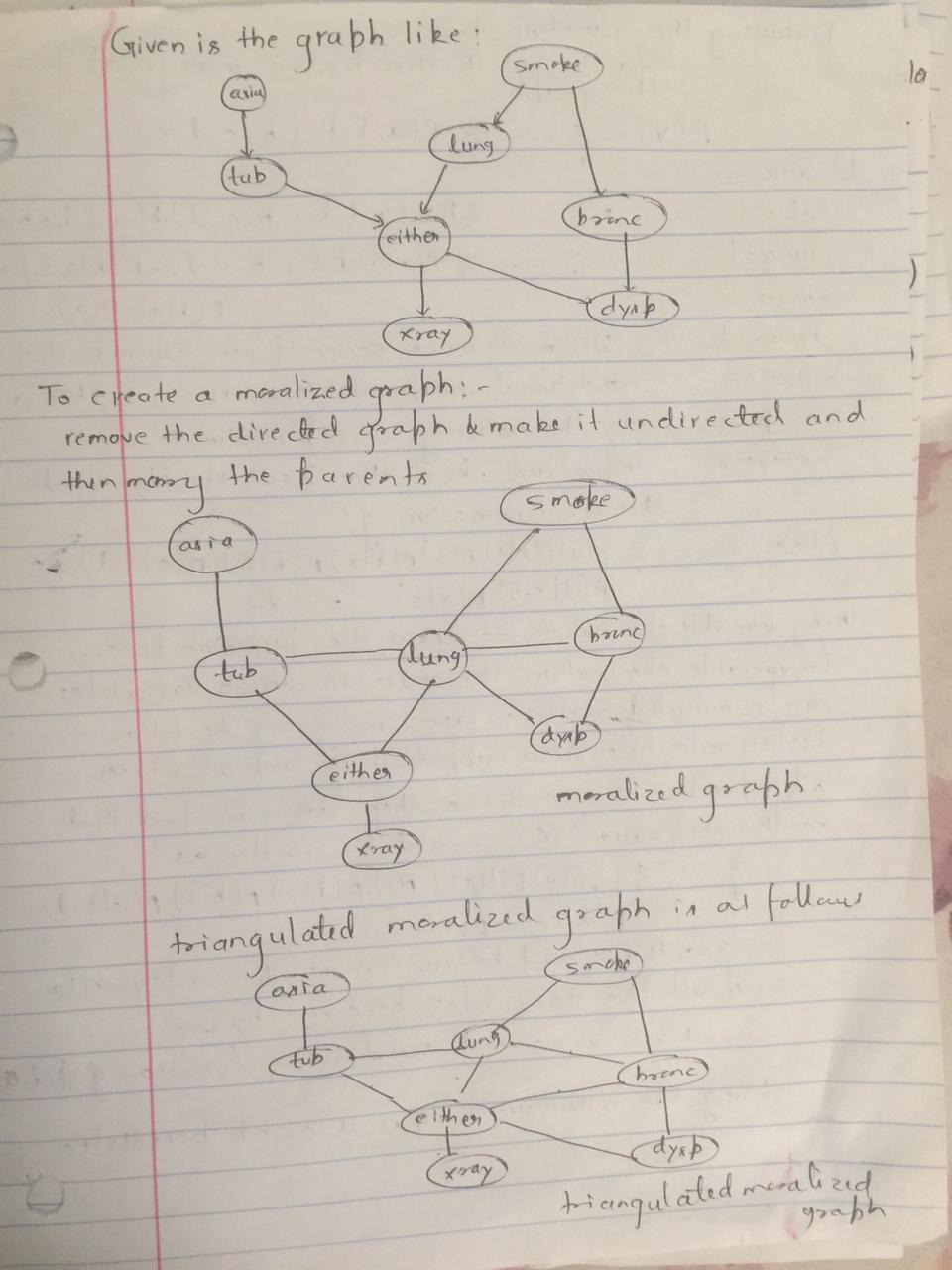
$ : chr [1:3] "either" "lung" "tub"

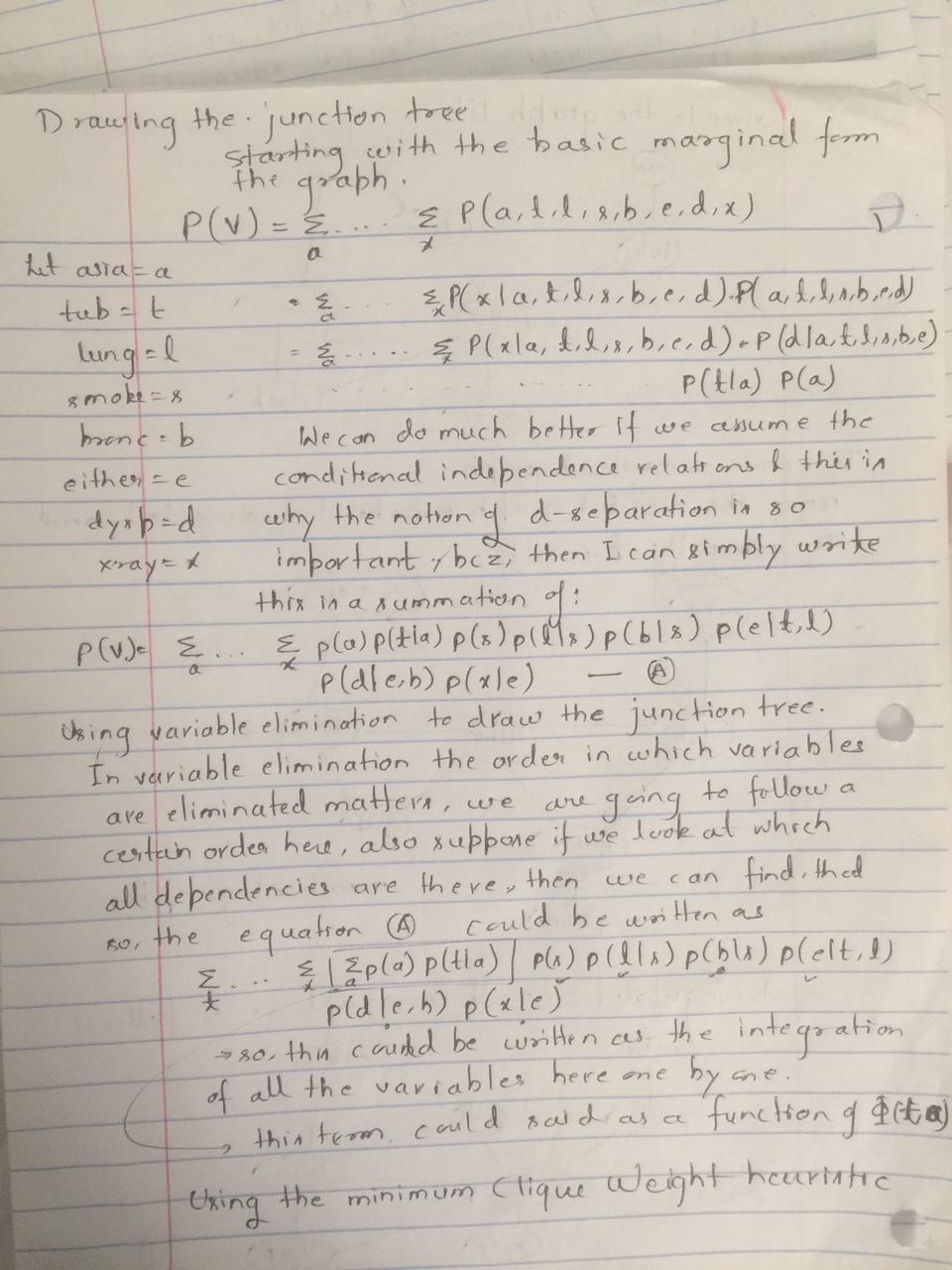
$ : chr [1:3] "either" "lung" "bronc"

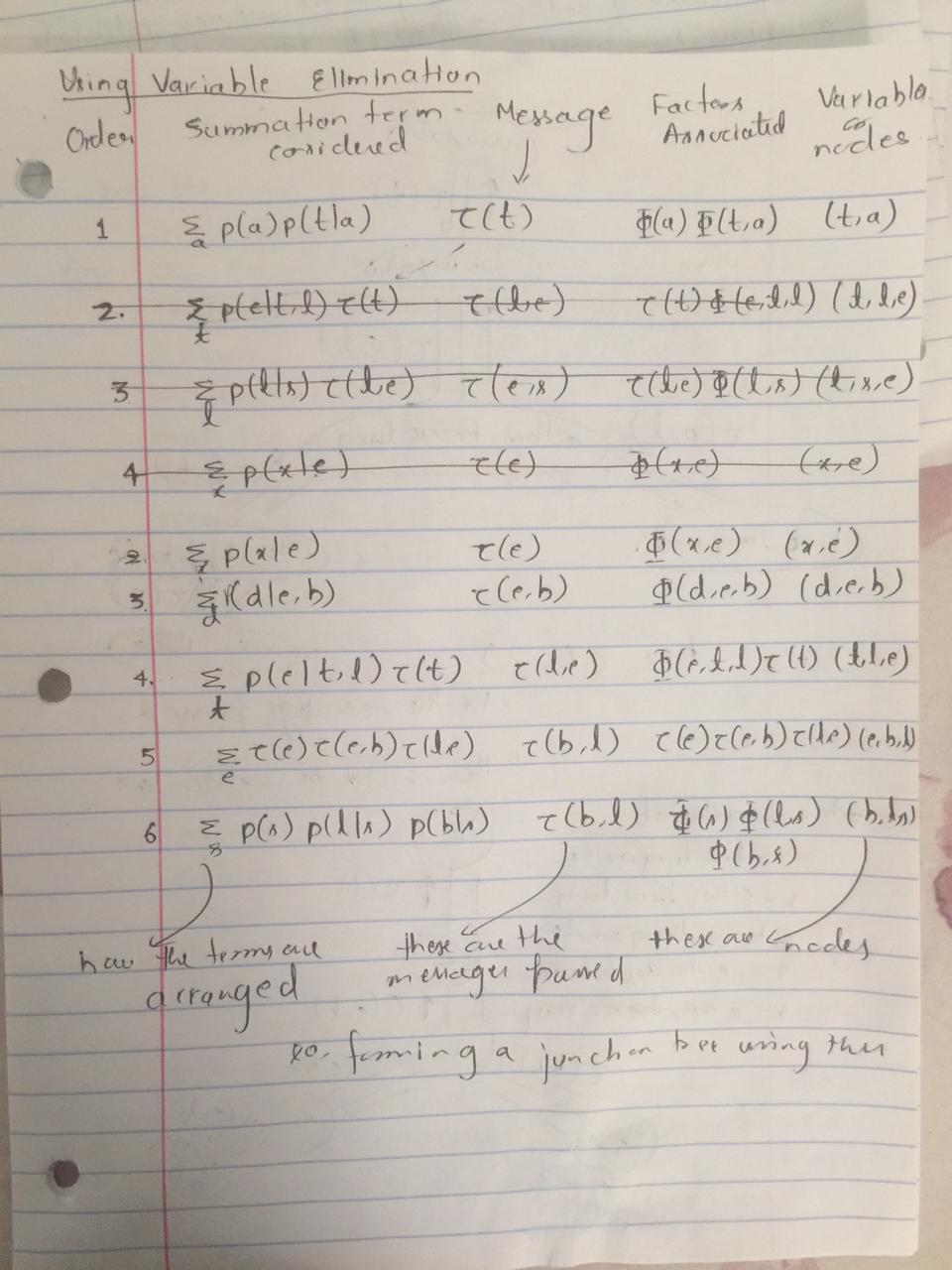
$ : chr [1:3] "smoke" "lung" "bronc"

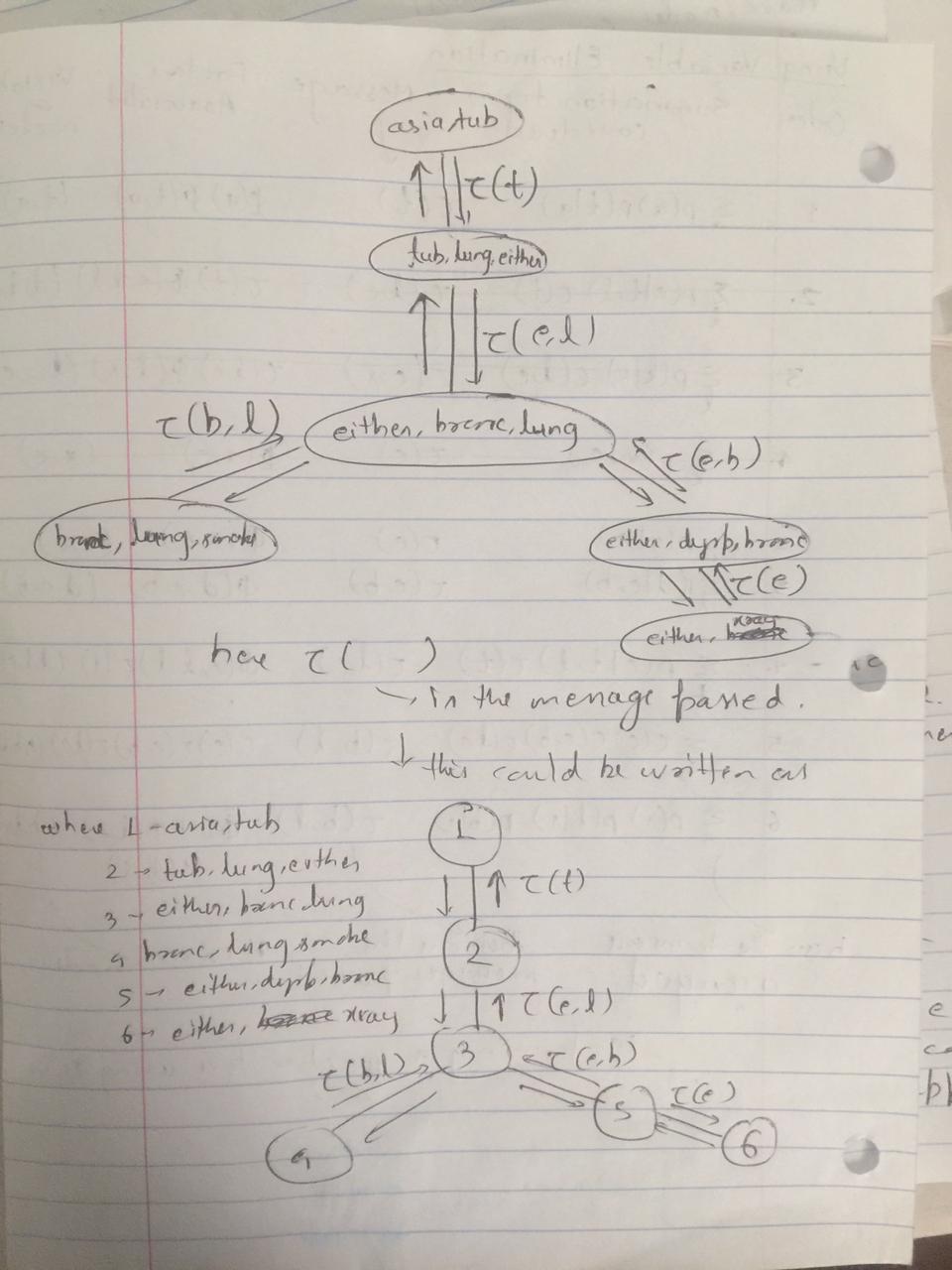
$ : chr [1:3] "either" "dysp" "bronc"

$ : chr [1:2] "either" "xray"





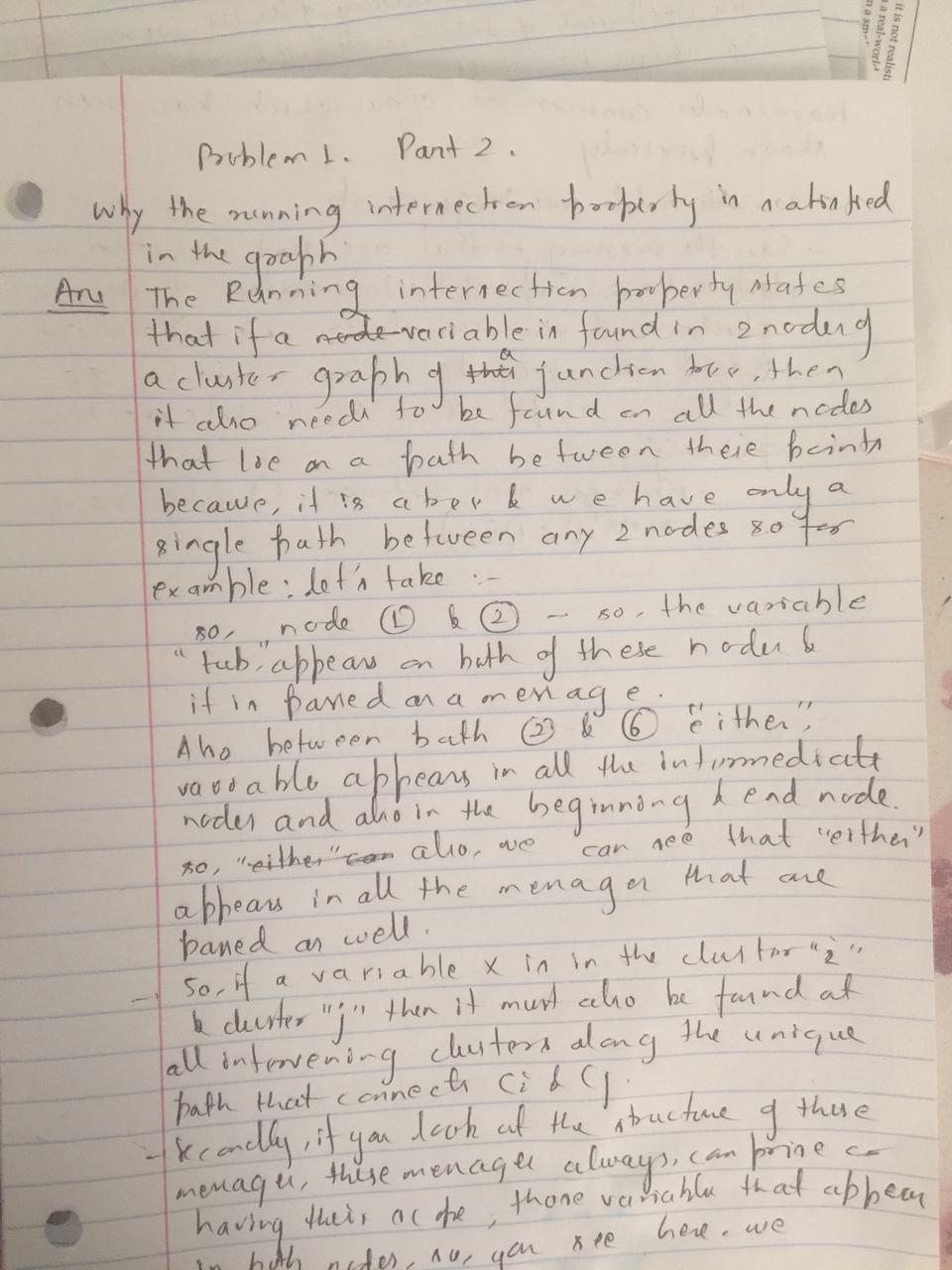


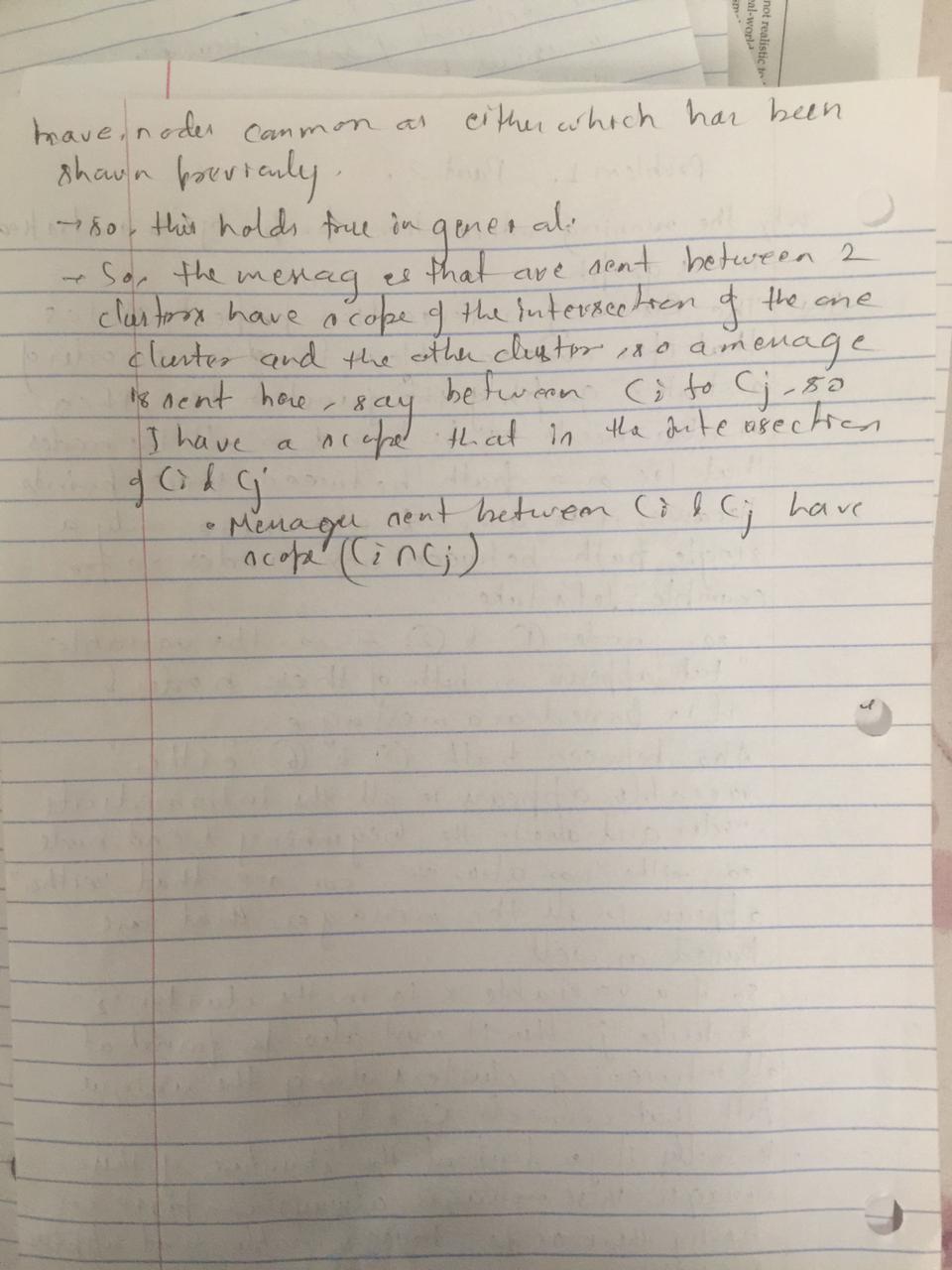


Problem 1 part2: Explain why the "running intersection property" is satisfied in your junction tree.

Solution part2

The Running Intersection Property is satisfied in the junction tree because it states that if a variable is found in 2 nodes of a cluster graph of a junction tree then it also needs to be found on all the nodes that lie on a path between these points because, it is a tree we have a single path between any 2 nodes. This has been explained below:

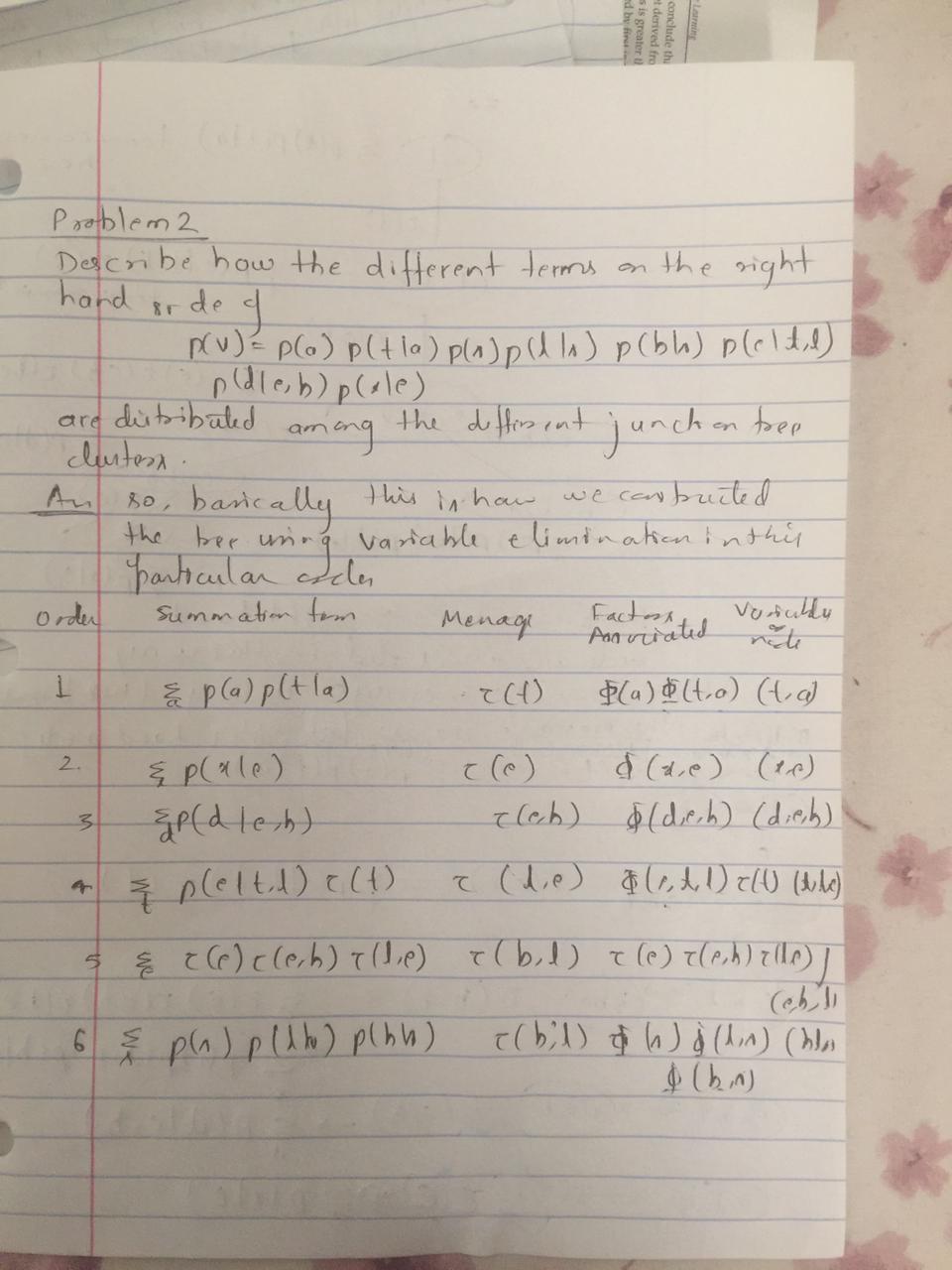


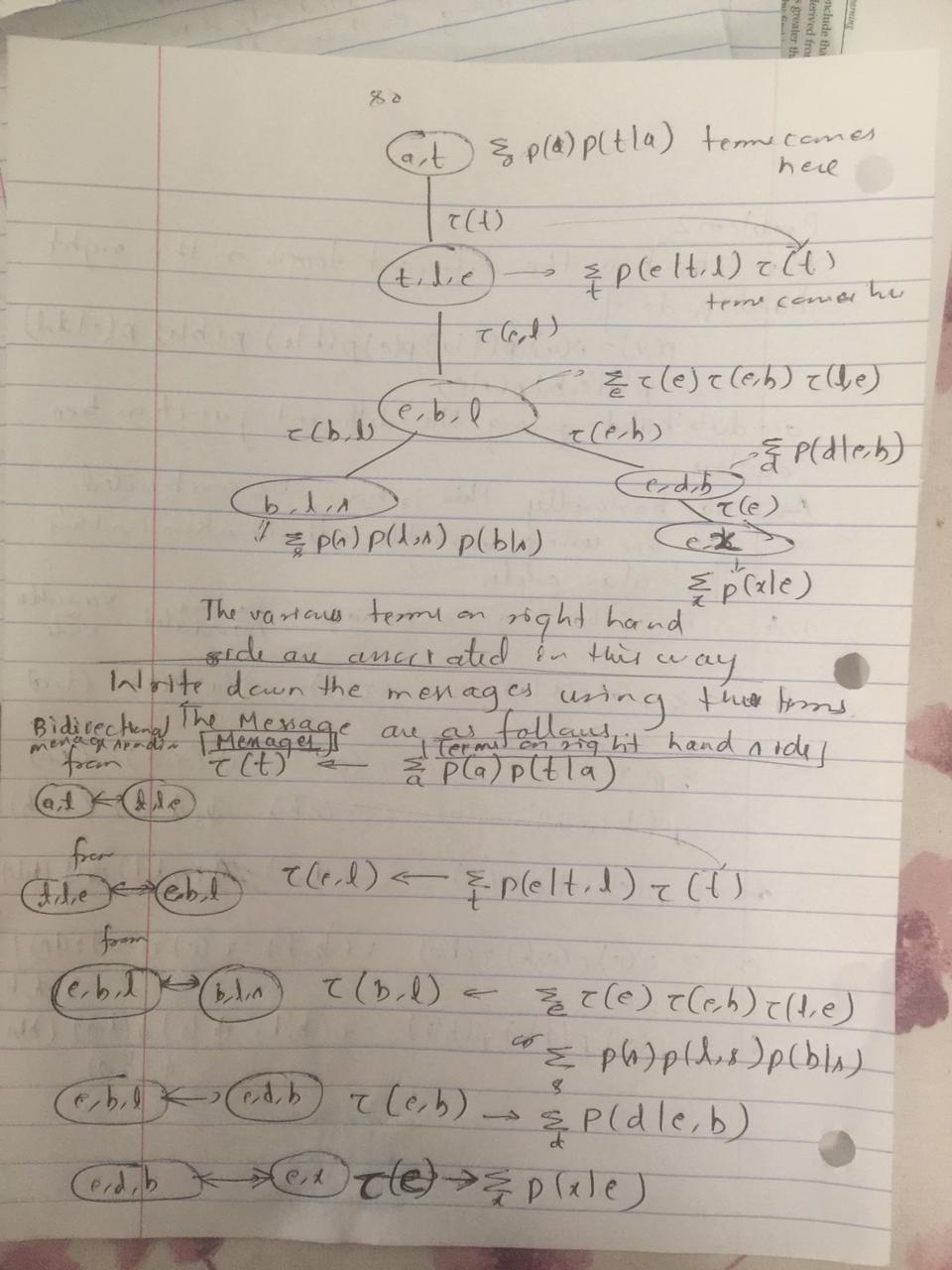


**Problem 2: Describe how the different terms on the right hand side of "p(V ) = p(a)p(t | a)p(s)p(l | s)p(b | s)p(e | t, l)p(d | e, b)p(x | e)" are distributed among the different juction tree clusters. Write out the messages using these terms and verify that the message passing algorithm indeed gives the cluster marginals.**

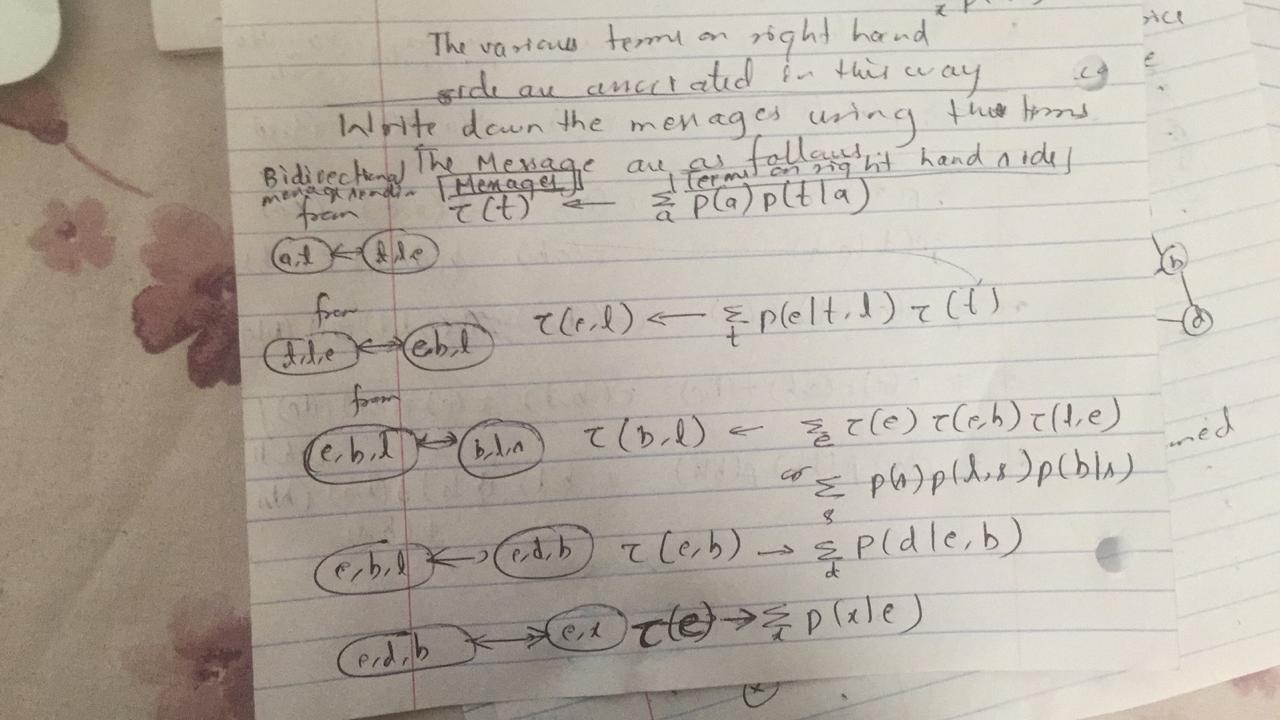
**Describe how the different terms on the right hand side of "p(V ) = p(a)p(t | a)p(s)p(l | s)p(b | s)p(e | t, l)p(d | e, b)p(x | e)" are distributed among the different juction tree clusters.**

**Solution Part 1:**

****

****

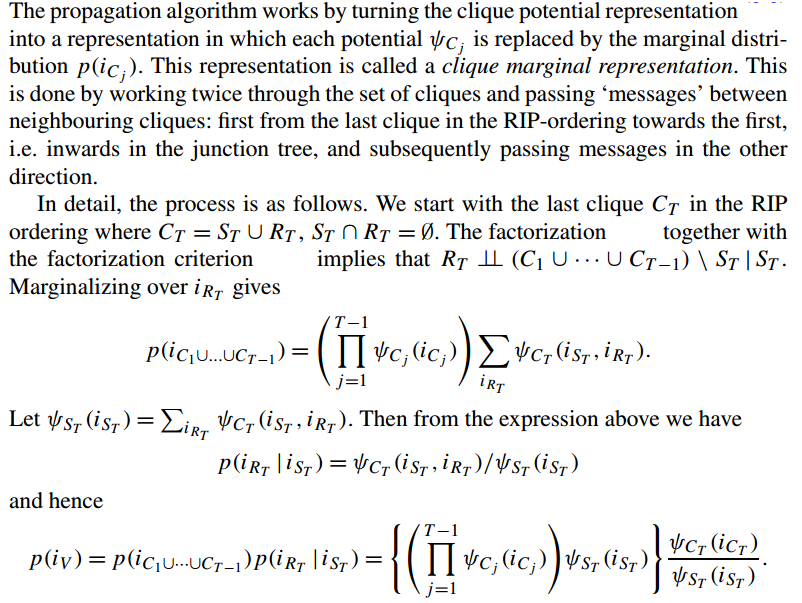
**Write out the messages using these terms**

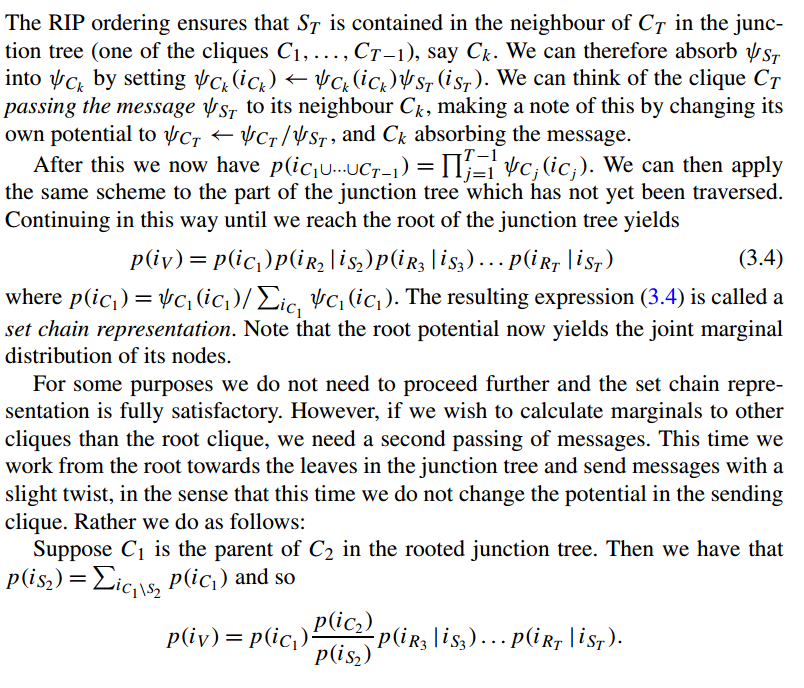
****

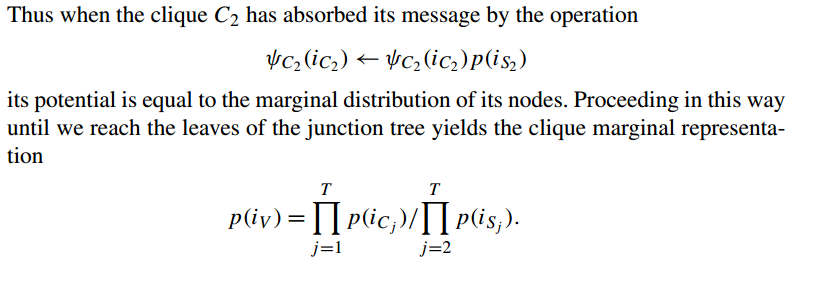
**Verify that the message passing algorithm indeed gives the cluster marginals.**

**Soln:** To be able to answer queries, the grain object must be propagated, which means that the clique potentials must be calibrated (adjusted) to each other. Propagation is done with the propagate() method for grain object

grn1c <- propagate(grn1c)



****

****

Hence proved and verified mathematically that message passing algorithm indeed gives the cluster marginals.

Problem:

Use message-passing algorithm to find the joint probability of "tub=yes, lung=yes, bronc=yes", given evidence that "asia=yes, xray=yes"?

Ans: We will use the junction tree constructed to find the joint probability of "tub=yes, lung=yes, bronc=yes", given evidence that "asia=yes, xray=yes".

Evidence is entered with setFinding() which creates a new grain object:

We set the evidence as asia=yes, xray=yes

gn1c.ev<setFinding(grn1c,nodes=c("asia","xray"),states=c("yes","yes"))

Then we can use the following query to find the joint probability of

"tub=yes, lung=yes, bronc=yes", given evidence that "asia=yes, xray=yes" which is gn1c.ev

querygrain(grn1c.ev,nodes=c("lung","bronc",”tub”), type="joint")

The answer is : 0.01063804

However, if it is known beforehand that interest will be in the joint distribution of a specific set U of variables, one can ensure that the set U is contained in a single clique in the triangulated graph. This can for example be done by first moralizing, then adding edges between all nodes in U, and finally triangulating the resulting graph. The price to be paid is that the cliques may become larger and since computational complexity is exponential in the largest clique, this can be prohibitive.

But it would give us the graph that we need:

To do this in practice we first need to compile the grain again

grn1c2 <- compile(grn1, root=c("lung", "bronc", "tub"),

propagate=TRUE)

grn1c2.ev<setFinding(grn1c2,nodes=c("asia","xray"),states=c("yes","yes"))

querygrain(grn1c2.ev,nodes=c("tub","bronc","lung"), type="joint")

output is same: 0.01063804

But the process is much faster.

system.time({for (i in 1:50)

+ querygrain(grn1c.ev,nodes=c("lung","bronc","tub"),

+ type="joint")

+ })

user system elapsed

1.5 0.0 1.5

system.time({for (i in 1:50)

+ querygrain(grn1c2.ev,nodes=c("lung","bronc","tub"),

+ type="joint")

+ })

user system elapsed

0.02 0.00 0.01

As we see the second step is much faster.

A detailed explanation is below:

