B Mesenterica Alpha

Table of contents

Introduction

Bondarzewia Mesenterica is a large beautiful and rare fungus that fruits in the Pacific Northwest. The growth of this fungus depends on a variety of environmental conditions. The 'Survey and Manage Species Project, USDA Forest Service, Northwest Forest Plan, Pacific Northwest Regional Office' has collected data on the presence of this fungus in a certain region along with the environmental conditions of that region. It is also observed that certain conditions directly affect the presence of this fungus and certain conditions indirectly affects it.

Given this domain information, we would like to ask questions like

- What is the chance of a certain region being a habitat for the fungus, given the environmental conditions in that region ?
- What are the likely environmental conditions of a certain region, given that the fungus grows there ?

We will build a Bayesian probabilistic model (a "Bayesian network") that fully encodes the domain information and, then, query the network to answer probabilistic questions of the kind mentioned above.

We will first construct the probabilistic model, implement the model in Netica, and then query the Netica network to answer our probabilistic queries.

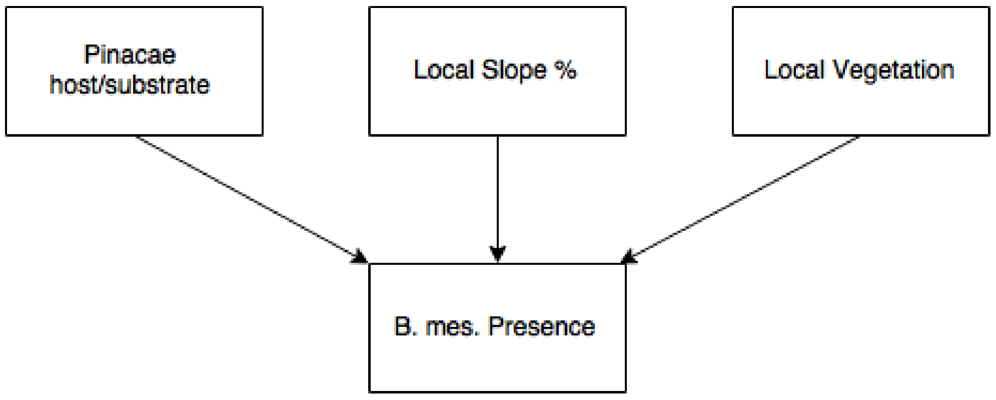
Building the Bayesian network for the growth of Bondarzewia Mesenterica fungus

A Bayesian network encodes probabilistic domain knowledge. Below, we will build the network in a hierarchical fashion, first identifying the factors that directly influence the growth of the fungus and then adding further factors that influence the direct factors.

Step 1: Modelling the direct factors

Based on the data collected it is observed that the presence of the fungus directly depends on 3 main factors:
1) The Pinacae host/substrate being adequate or inadequate
2) The Local Slope
3) The amount of Local Vegetation

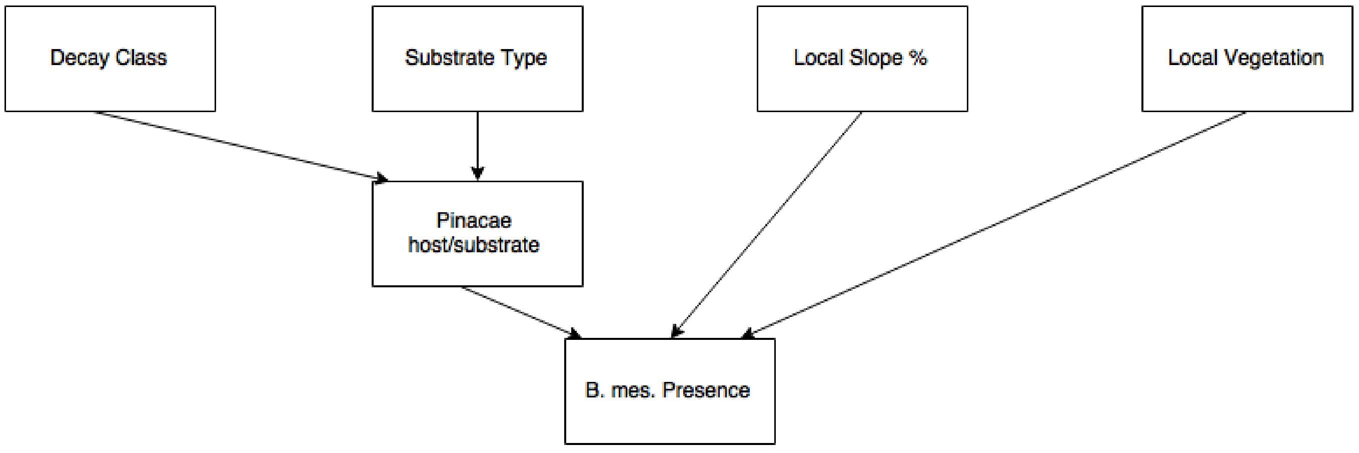
Hence we connect these 3 nodes to the first node which indicates the presence of the B. mes fungus



Step 2: Modelling indirect factors

It is also observed that the amount of Pinacae host/substrate depends on 2 factors:
1) Decay Class
2) Substrate Type

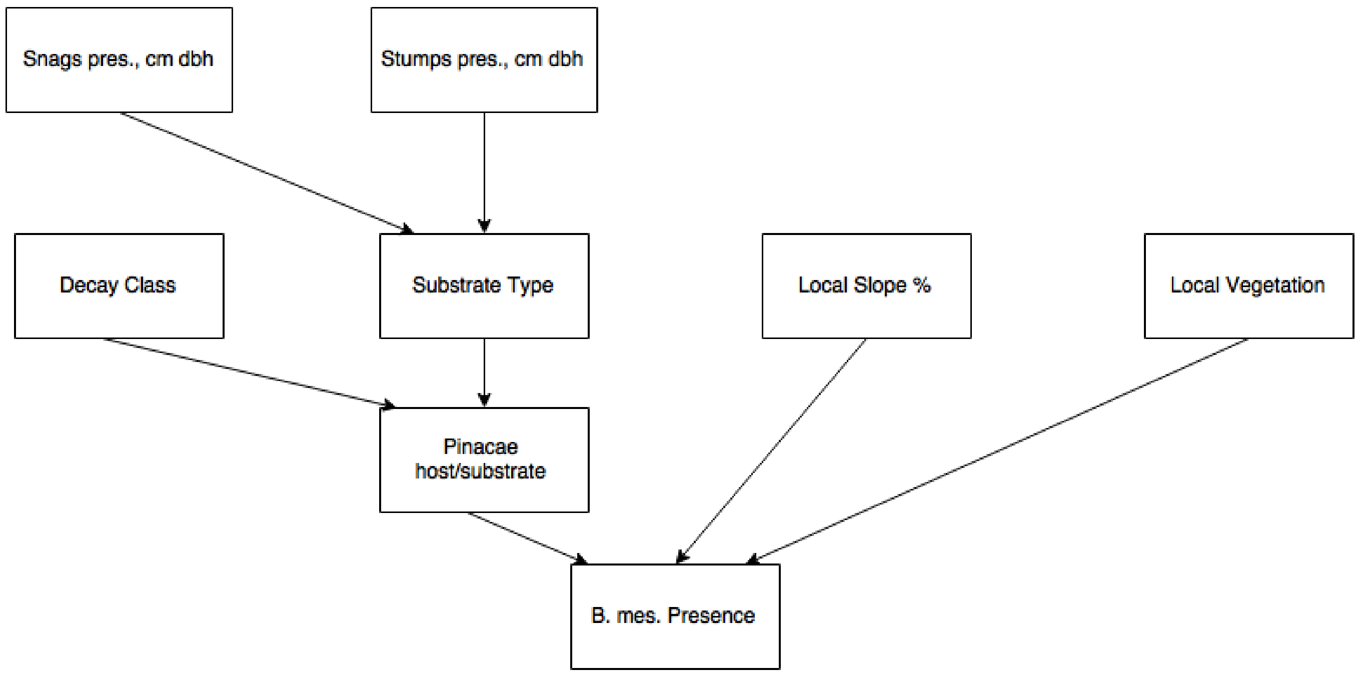
We thus add these two nodes in connection with the Pinacae host/substrate node



Step 3: Modelling indirect factors

The Substrate Type further depends on:
1) The amount of Snags present
2) The amount of Stumps present

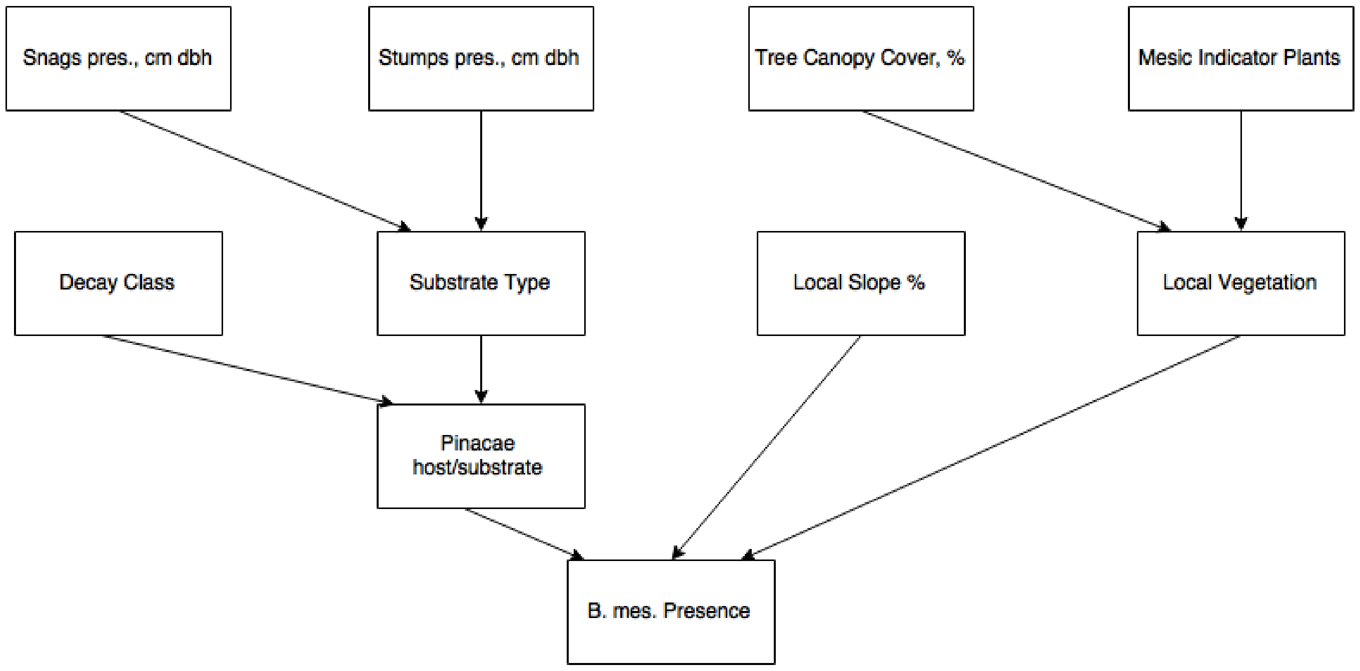
We connect these two nodes to the node Substrate Type



Step 4: Modelling indirect factors

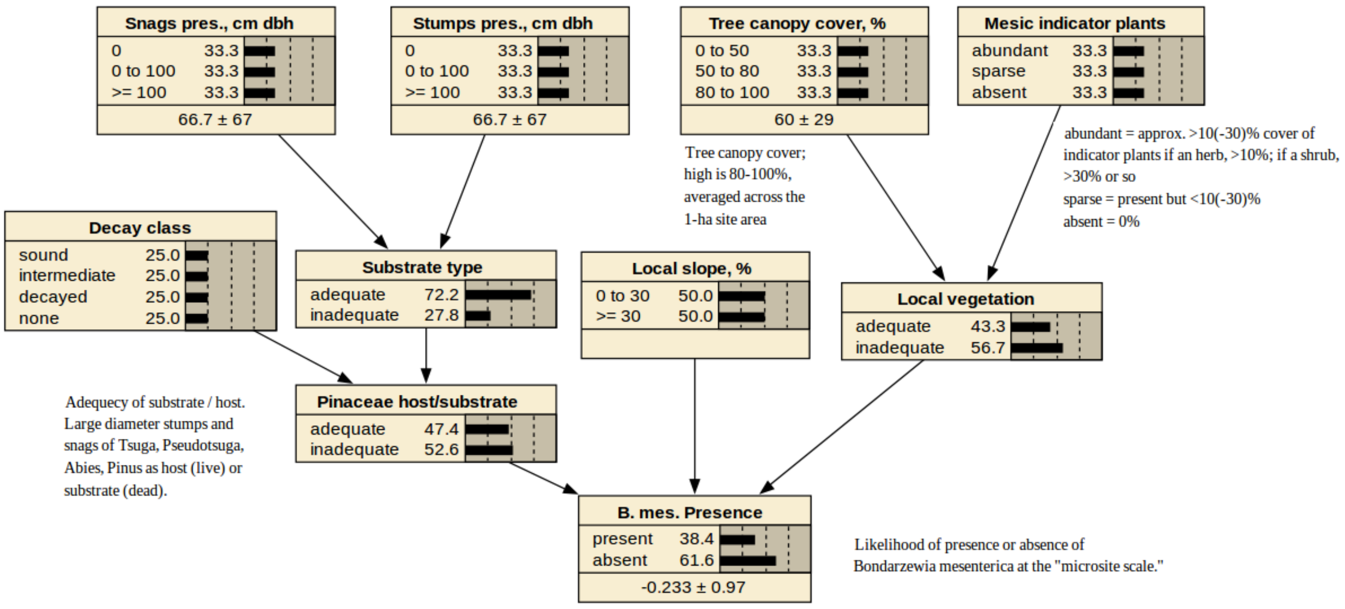
Finally the Local Vegetation of that region depends on:
1) The Tree Canopy Cover of that region
2) Mesic indicator plants

Again, we connect these nodes to the Local Vegetation



Step 5: Adding quantitative information - conditional probability tables

Now we will incorporate the probabilities of each node into the network based on the data collected. The resulting network is shown below.



This model is a Microsite Scale BBN Model for the Fungus Bondarzewia Mesenterica and it shows the relation of a certain entity with another entity. Each block is called a 'node'. A node denotes a certain condition and each node has a few 'states'. For example, the node 'Local Vegetation' has the states 'adequate' and 'inadequate' and they denote the probability of the amount local vegetation being adequate or inadequate given a set of conditions.
The state of a particular node can affect other nodes as well. This is represented using arrows and are reffered to as the connections between the nodes. For example, the 'Tree Canopy Cover' node is connected to 'Local Vegetation', which means that the amount of local vegetation depends to a certain extent on the tree canopy cover of that region.

Implementing the network using Netica

The Bayesian network described above will now be implemented in Netica, following the steps below.

Create a Netica object

First the Netica object needs to be created. This object will be used throughout the program and we can make use of the Netica API using this. All the nodes, connections etc. that need to be created on a network can be done using this.

In [1]:

```
from NeticaPy import Netica



# Creating the Netica object
N = Netica()
```

It would be good to also print out the success(or failure) message returned by the Netica API after the object has been created. Before that, we also need to create a working environment for the network. We create this enviromiment in a variable `env`

In [2]:

```
# Create a bytearray variable that will store messages
# that are returned by the Netica API
mesg = bytearray()

# Create a Netica environment
env = N.NewNeticaEnviron_ns("", None, "")
res = N.InitNetica2_bn (env, mesg)

# Printing the success message
print mesg
```

```
Netica 5.04 Linux (AF), (C) 1992-2012 Norsys Software Corp.

Netica operating without a password; there are some limitations.
```

Initializing the Bayesian Network

Now that the Netica object has been created we need to start building the Bayesian network. Hence, a `bayesian_network` variable needs to be initialized by providing a name to the network and the environment. We use `Mesentria` as the name of the network and use the `env` environment we created earlier.
This `bayesian_network` will hold the definition of the entire network including the nodes, probabilities, how they are connected etc.

In [3]:

```
bayesian_network = N.NewNet_bn ("Mesentria", env)
```

Also create a variable `INFINITY_ns` which can be used later to specify values/ranges that go to infinity. For example, if you see the node 'Stumps Pres., cm dbh', the last state is >=100 which means that the values go till infinity. We will be therefore be using the `INFINITY_ns` variable in such cases.

In [4]:

```
INFINITY_ns = N.GetInfinityDbl_ns()
```

Defining nodes for the network

Create each node by providing:

1) Name of node
2) Number of states in the node
3) The network it belongs

For example, to define the node 'Snags pres., cm dbh' shown in the model, we use the variable `Snags` to store the node. The node is named as 'Snags', having 3 states and belonging to the network `bayesian_network`

In [5]:

```
# node for Snags pres., cm dbh having 3 states
Snags = N.NewNode_bn("Snags", 3, bayesian_network)

# node for Stumps pres., cm dbh having 3 states
Stumps = N.NewNode_bn("Stumps", 3, bayesian_network)

# node for Tree Canopy Cover % having 3 states
TreeCanopyCover = N.NewNode_bn("TreeCanopyCover", 3, bayesian_network)

# node for Mesic Indicator Plants having 3 states
MesicIndicatorPlants = N.NewNode_bn("MesicIndicatorPlants", 3, bayesian_ne
twork)

# node for Decay Class having 4 states
DecayClass = N.NewNode_bn("DecayClass", 4, bayesian_network)

# node for Substrate Type having 2 states
SubstrateType = N.NewNode_bn("SubstrateType", 2, bayesian_network)

# node for Local Scope, % having 2 states
LocalSlope = N.NewNode_bn("LocalSlope", 2, bayesian_network)

# node for Local Vegetation having 2 states
LocalVegetation = N.NewNode_bn("LocalVegetation", 2, bayesian_network)

# node for Pinaceae Host/Substrate having 2 states
Pinaceae = N.NewNode_bn("Pinaceae", 2, bayesian_network)

# node for B.mes. presence having 2 states
BMesPresence = N.NewNode_bn("BMesPresence", 2, bayesian_network)
```

Set node levels for Continuous variables

If you look at the model you will see that there are some nodes whose states are continuous variables that have been made discrete. For such variables, we need to set the node levels before assigning names to the states.

For example, look at the node 'Local Slope %' in the model. It is continuous but has been made discrete. Before assigning names to its states, we need to divide its infinite range into 2 parts which are `0 to 30` and `>=30`
We do this by using the method `SetNodeLevels_bnL()` and specifying 3 things:
1) Node
2) Number of states
3) Values where the levels need to be set

In [6]:

```
# Setting node levels for Snags pres., cm dbh
N.SetNodeLevels_bn(Snags,3,[0,0,100,INFINITY_ns])

# Setting node levels for Stumps pres., cm dbh
N.SetNodeLevels_bn(Stumps,3,[0,0,100,INFINITY_ns])

# Setting node levels for Snags pres., cm dbh
N.SetNodeLevels_bn(TreeCanopyCover,3,[0,50,80,100])

# Setting node levels for Snags pres., cm dbh
N.SetNodeLevels_bn(LocalSlope,2,[0,30,INFINITY_ns])
```

Set state names for the nodes

Set the names for the states in each of the nodes. State names can be set using the `SetNodeStateNames_bn() method` and providing 2 parameters:
1) Node
2) A string containing the names of the states separated by comma

(Netica doesn't allow state names to begin with a number so we have used 'r' to denote 'range' and 'greater' to denote 'greater than' at the beginning of the name)

In [7]:

```
N.SetNodeStateNames_bn(Snags, "r0,r0_100,greater100")
N.SetNodeStateNames_bn(Stumps, "r0,r0_100,greater100")
N.SetNodeStateNames_bn(TreeCanopyCover, "r0_50,r50_80,r80_100")
N.SetNodeStateNames_bn(LocalSlope, "r0_30,greater30")
N.SetNodeStateNames_bn(MesicIndicatorPlants, "abundant,sparse,absent")
N.SetNodeStateNames_bn(DecayClass, "sound,intermediate,decayed,none")
N.SetNodeStateNames_bn(SubstrateType, "adequate,inadequate")
N.SetNodeStateNames_bn(LocalVegetation, "adequate,inadequate")
N.SetNodeStateNames_bn(Pinaceae, "adequate,inadequate")
N.SetNodeStateNames_bn(BMesPresence, "present,absent")
```

Connecting the nodes

Now that the nodes have defined, we need to specify connections between the nodes. These are indicated by arrows in the model diagram.
Here the AddLink_bn() method is used which accepts two nodes as parameters and adds a link between them. The comments in the code below indicate which 2 nodes are being connected

In [8]:

```
# Stumps pres., cm dbh is connected to Substrate Type
N.AddLink_bn(Stumps, SubstrateType)

# Snags pres., cm dbh is connected to Substrate Type
N.AddLink_bn(Snags, SubstrateType)

# Snags Mesic indicator plants is connected to Local Vegetation
N.AddLink_bn(MesicIndicatorPlants, LocalVegetation)

# Tree canopy cover is conntected to Local Vegetation
N.AddLink_bn(TreeCanopyCover, LocalVegetation)

# Substrate Type is connected to Pinaceae Host/Substrate
N.AddLink_bn(SubstrateType, Pinaceae)

# Decay Class is connected to Pinaceae Host/Substrate
N.AddLink_bn(DecayClass, Pinaceae)

# Pinaceae Host/Substrate is connected to B.mes. presence
N.AddLink_bn(Pinaceae, BMesPresence)

# Local Vegetation is connected to B.mes. presence
N.AddLink_bn(LocalVegetation, BMesPresence)

# Local Slope is connected to B.mes. presence
N.AddLink_bn(LocalSlope, BMesPresence)
```

Out[8]:

```
2
```

Specifying probabilities at each node

The probabilities of each node need to be specified before compiling the network. The `SetNodeProbs()` method is used to do this. This method requires you to specify the node followed by the name/values for the states.

(For this particular example we have stored the probabilities in a CSV file in the folder 'data'. This is however not required as you can directly enter the probabilities into the program if they are small in quantity)

1) Set the probabilites for the node 'Substrate Type'

In [9]:

```
substrate_type_probs = []
# collect the probabilities from the CSV file
with open('data/mesenteria/substrate_type.csv') as f:
    for row in f.readlines():
        substrate_type_probs.append(row.strip('\n').split(','))

substrate_map = {'0':'r0', '0-100':'r0_100', '>100':'greater100'}
for row in substrate_type_probs:
    N.SetNodeProbs(SubstrateType,substrate_map[row[0]],substrate_map[row
[1]],float(row[2]),float(row[3]))
```

2) Set the probabilities for the node 'Local Vegetation'

In [10]:

```
local_vegetation_probs = []
# collect the probabilities from the CSV file
with open('data/mesenteria/local_vegetation.csv') as f:
    for row in f.readlines():
        local_vegetation_probs.append(row.strip('\n').split(','))

substrate_map = {'0-50':'r0_50', '50-80':'r50_80', '80-100':'r80_100'}
for row in local_vegetation_probs:
    N.SetNodeProbs (LocalVegetation,row[0],substrate_map[row[1]],float(row
[2]),float(row[3]))
```

3) Set the probabilities for the node Pinaceae Host/Substrate

In [11]:

```
local_vegetation_probs = []
# collect the probabilities from the CSV file
with open('data/mesenteria/pinaceae.csv') as f:
    for row in f.readlines():
        local_vegetation_probs.append(row.strip('\n').split(','))

substrate_map = {'0-50':'r0_50', '50-80':'r50_80', '80-100':'r80_100'}
for row in local_vegetation_probs:
    N.SetNodeProbs (Pinaceae,row[0],row[1],float(row[2]),float(row[3]))
```

4) Set the probabilities for the node B.mes. presence

In [12]:

```
bmes_presence_probs = []
# collect the probabilities from the CSV file
with open('data/mesenteria/bmes_presence.csv') as f:
    for row in f.readlines():
        bmes_presence_probs.append(row.strip('\n').split(','))

substrate_map = {'0-30':'r0_30', '>30':'greater30'}
for row in bmes_presence_probs:
    N.SetNodeProbs (BMesPresence,row[0],row[1],substrate_map[row[2]],float
(row[3]), float(row[4]))
```

Checking for errors occured

The ErrorMessage_ns() method is used to check if any error occured in the Netica API. You need to fix your program based on the error message you receive. If you don't get any message then it means that everything is fine and you can go ahead with compiling the network.

In [13]:

```
# print the error message in case of any errors within Netica
print N.ErrorMessage_ns(N.GetError_ns(N, 5, 0))
```

Compiling the network

The compilation step transforms the internal representation of the network into a more computationally efficient form. This step, though essential, is transparent to the user. We use the `CompileNet_bn()` method to complie a network by providing the network variable as a parameter.

In [14]:

```
N.CompileNet_bn(bayesian_network)
```

Testing the network with known outcomes:

In order to check if the network has been set up properly, we can try getting the belief for the conditions of B. Mes. Presence being 'absent' or 'present', since the result of that is already shown in the model.
The `GetNodeBelief()` method calculates the probability of a certain state. The code below prints the probability of the node 'B. mes. Presence' having a state absent. In other words it prints out the probability of the fungus being absent for the current set of environmental conditions.

In [15]:

```
belief = N.GetNodeBelief("BMesPresence", "absent", bayesian_network)
print """The probability is %g"""% belief
```

```
The probability is 0.61628
```

The code below prints the probability of the node 'B. mes. Presence' having a state present. In other words it prints out the probability of the fungus being present for the current set of environmental conditions.

In [16]:

```
belief = N.GetNodeBelief("BMesPresence", "present", bayesian_network)
print """The probability is %g"""% belief
```

```
The probability is 0.38372
```

Querying the network for probabilistic answers

Now that the network is ready, we can use it to get the probabilities of certain states given a set of findings. The method `EnterFinding()` is used to specify findings. It takes 3 parameters:
1) Name of the node
2) State of that node
3) The network variable(`bayesian_network`)

In simple words, a 'finding' is the state of a node that you are sure of.

Case 1:

Getting the probability of B. Mes. Presence being present if it is known that SubstrateType is 'adequate'

In [17]:

```
N.EnterFinding("SubstrateType", 'adequate', bayesian_network)
belief = N.GetNodeBelief ("BMesPresence", "present", bayesian_network)
print """The probability that B. Mes. is present is %g"""% belief
```

```
The probability that B. Mes. is present is 0.438083
```

Case 2:

Getting the probability of Pinaceae host/substrate being 'adequate' given that B. Mes. Presence is 'present' and 'decay class' is decayed.

In [18]:

```
N.EnterFinding ("BMesPresence", "present", bayesian_network)
N.EnterFinding ("LocalVegetation", "adequate", bayesian_network)
belief = N.GetNodeBelief ("Pinaceae", "adequate", bayesian_network)
print """The probability of Pinaceae host/substrate being 'adequate' is %
g"""% belief
```

```
The probability of Pinaceae host/substrate being 'adequate' is 0.93617
```

Closing the program

Once all probabilistic queries are answered, the network has to be closed. This will de-allocate the memory that had been allocated when the Netica object was initialized.

We use the `DeleteNet_bn()` to delete the Bayesian Network we had initially created and use the method `CloseNetica_bn()` to close Netica

In [19]:

```
N.DeleteNet_bn(bayesian_network)
res = N.CloseNetica_bn (env, mesg)

print mesg
```

```
Leaving Netica.
```